UNIVERSIDADE DA CORUÑA

ESCUELA UNIVERSITARIA POLITÉCNICA

Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO DE FIN DE GRADO

TFG Nº: **770G01A52**

TÍTULO: **DESARROLLO Y TESTEO DE UN EMULADOR DE PLANTAS INDUSTRIALES BASADO EN ARDUINO**

AUTOR: **Esteban Velo Sánchez**

TUTORES: **José Luis Calvo Rolle**
José Luis Casteleiro Roca

FECHA: **JUNIO DE 2014**

Fdo.: EL AUTOR

Fdo.: EL TUTOR

ÍNDICE GENERAL

Índice de figuras

Índice de tablas

MEMORIA

TÍTULO: **DESARROLLO Y TESTEO DE UN EMULADOR DE PLANTAS INDUSTRIALES BASADO EN ARDUINO**

MEMORIA

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

15405 - FERROL

DATA: JUNIO DE 2014

AUTOR: ESTEBAN VELO SÁNCHEZ

Fdo.:

Capítulo 1

Objeto

El objeto del presente proyecto es el diseño e implementación de un dispositivo autónomo capaz de simular el comportamiento de plantas industriales mediante la emulación de su función de transferencia. El dispositivo tendrá como finalidad la realización de prácticas de laboratorio ó ensayos industriales basados en tarjetas de adquisición de datos sin la necesidad de disponer de plantas reales, es decir, realizar pruebas e ingeniería de control sobre una planta virtual (emulada).

El equipo estará perfectamente indicado para su utilización con tarjetas de adquisición de datos asequibles como aquellas formadas por un Arduino y el software de Matlab. También podrá ser utilizado por tarjetas de adquisición de datos profesionales que normalmente tienen un coste mucho más elevado.

De esta forma se trata de desarrollar y testear, un emulador de funciones de transferencia continuas (aproximadas) ó funciones de transferencia discretas a las que se permitirá configurar los parámetros principales de la función de transferencia así como el periodo de muestreo de la misma cuando proceda.

Capítulo 2

Alcance

El alcance del presente proyecto debe cubrir todas las etapas de aprendizaje, diseño, implementación y testeo del equipo que se pretende diseñar. De esta forma, el alcance del presente proyecto se puede dividir en tres bloques principales claramente diferenciados:

1. Aprendizaje de Arduino sobre Matlab como tarjeta de adquisición de datos.

2. Diseño e implementación del emulador de funciones de transferencia SISO (Single Input Single Output, una entrada y una salida).

3. Pruebas de funcionalidad y testeo del equipo diseñado.

El primer bloque está destinado a la posterior utilización y testeo del emulador de funciones de transferencia. Se trata de emplear el hardware de Arduino como tarjeta de adquisición de datos sobre el software de Matlab. De esta forma se consigue una tarjeta de adquisición de datos asequible que permite utilizar el dispositivo emulador y, a la vez, incluye la posibilidad realizar un control mediante el software de Matlab sobre la función de transferencia emulada. De esta forma se podrá observar como funciona un determinado control sobre una función de transferencia determinada que puede o no corresponderse con una planta real.

El segundo bloque será el cuerpo principal del presente proyecto en el que se diseñará el emulador de funciones de transferencia SISO. El dispositivo deberá incluir diferentes opciones en cuanto a la función de transferencia y deberá, al menos, contener las siguientes características que permitirán al dispositivo ser una herramienta útil a la hora de simular el comportamiento de determinadas funciones de transferencia reales ó de ámbito didáctico:

- Simulación de funciones de transferencia discretas y continuas (aproximadas) SISO.
- Incluir una serie de funciones de transferencia tipo (SISO) asociadas a sistemas de referencia en control.
- Posibilidad de introducir nuevas funciones de transferencia.
- Deberá permitir modificar los parámetros principales de las funciones de transferencia.
- Posibilidad de modificar el periodo de muestreo de las funciones de transferencia discretas.
- Interfaz teclado y LCD (Liquid Crystal Display, pantalla de cristal líquido) para poder interactuar con el dispositivo.
- Funcionamiento autónomo del dispositivo.

El equipo se implementará en una caja con fuente de alimentación de modo que el funcionamiento del mismo tan sólo dependa de la corriente eléctrica. La implementación deberá realizarse de modo que respete los principios básicos de ergonomía y que le brinde una durabilidad elevada al equipo.

En cuanto al tercer bloque cabe comentar que se deberán comprobar todas y cada una de las opciones que incorpora el dispositivo emulador. Además deberá comprobarse el correcto funcionamiento del dispositivo de manera que los resultados obtenidos con el mismo sean idénticos (ó muy aproximados) a los que se obtendrían en un programa de simulación como puede ser Matlab. Deben realizarse múltiples pruebas con configuraciones diferentes que demuestren el buen funcionamiento del equipo bajo distintas situaciones. Se deben comprobar que los resultados obtenidos son los correctos y que el tiempo (periodo de muestreo) es constante y conocido por el usuario. Se deben comprobar también todas las funciones de transferencia que el equipo lleve preinstaladas.

Por otra parte el dispositivo se debe comportar como una planta real, es decir, tendrá una entrada y una salida. La entrada a la planta será la señal de control que se generará a partir de un regulador (por ejemplo PID). Este regulador tendrá como entrada la señal de error que se obtiene de la diferencia entre la consigna y la variable de proceso (la señal de salida de la planta, medida con la tarjeta de adquisición de datos. Con este sistema la tarjeta de adquisición de datos (que puede ser un Arduino) será la

encargada de realizar la realimentación y de esta forma cerrar el lazo de control.

Con este sistema la única diferencia con respecto a probar y controlar una planta real es que la misma no existe, es una planta virtual emulada por el equipo.

Capítulo 3

Antecedentes

3.1. La función de transferencia

En la teoría de control, a menudo se usan las funciones de transferencia para caracterizar las relaciones de entrada-salida de sistemas que se describen mediante ecuaciones diferenciales LTI (Linear Time-Invariant, lineal e invariante con el tiempo). De esta forma la función de trasferencia de un sistema LTI se define como el cociente entre la transformada de Laplace (también conocida como transformada S) de la salida y la transformada de Laplace de la entrada, bajo la suposición de que las condiciones iniciales son nulas. Esta definición sería válida para todos los sistemas continuos. Sin embargo, para los sistemas discretos bastaría con sustituir el término de transformada de Laplace por el de transformada Z. La función de transferencia estará formada entonces por un cociente entre ecuaciones algebraicas en S (ó en Z). Los sistemas realizables son aquellos en los que el grado del numerador es igual ó menor que el del denominador. En éstos se define el orden del sistema como el grado del polinomio del denominador.

Al conocer la función de transferencia de un sistema (ya sea continuo ó discreto) se puede conocer la forma en la que responderá ante un determinado estímulo de entrada,es decir, se conoce la dinámica del sistema. La función de transferencia es una propiedad de un sistema, independientemente de la magnitud ó naturaleza de la entrada que se aplique.

Existen métodos experimentales para obtener la función de transferencia de un sistema introduciendo entradas conocidas y estudiando la salida del sistema. Estos se denominan métodos de identificación como, por ejemplo, el método de mínimos cuadrados.

En el presente proyecto se tratarán las funciones de transferencia como algo conocido y, como ya se mencionó anteriormente, se incluirán una serie de funciones de transferencia tipo de sistemas reales que se tratarán de simular.

3.2. Emulación de funciones de transferencia

Se pueden encontrar emuladores de funciones de transferencia implementados anteriormente. Sin embargo, estos distan mucho de lo que se pretende realizar en el presente proyecto. En cuanto al software, se trata, unicamente, de emuladores de funciones de transferencia continuas (su aproximación realmente). Además, en muchos casos, las aproximaciones utilizadas no son las correctas ya que si no se emplea la transformación de Tustin (sección 10.1.2), un sistema inestable se puede convertir en estable (y viceversa) al realizar la aproximación. En el presente proyecto el emulador tendrá diversas posibilidades con las que se podrán simular tanto funciones de transferencia continuas como discretas.

En cuanto al hardware empleado existen diseños implementados con autómatas programables en los que se aprovecha su capacidad de cálculo para simular las funciones de transferencia. Sin embargo, un autómata tiene un precio bastante superior al de un Arduino ya que el primero tiene la robustez indicada para operar en un ámbito industrial. Existen otros emuladores más antiguos en los que el hardware utilizado estaba compuesto por circuitos de electrónica analógica con una complejidad elevada en los que también se podían simular una serie bastante limitada de funciones de transferencia continuas.

No se han encontrado referencias ni de un emulador físico de funciones de transferencia discretas ni mucho menos algo de este tipo sobre un Arduino. Lo más parecido que se puede encontrar son los programas de ordenador que permiten simular funciones de transferencia como pueden ser Matlab y su toolbox Simulink. Sin embargo estos tienen los siguientes inconvenientes:

- Necesidad de utilizar un ordenador para poder interactuar.
- Programación compleja a la hora de crear una aplicación facilmente interactiva.
- Imposibilidad de controlar la función de transferencia emulada con una señal de entrada externa.

- Las simulaciones no se pueden ver en tiempo real ó éste es muy limitado.

Se puede concluir que el equipo implementado será algo novedoso que incluirá diversas opciones que le brindarán una alta utilidad en los laboratorios de control tanto continuo como discreto, incluso en laboratorios de investigación y desarrollo ó ensayos de empresas del ámbito industrial.

Capítulo 4

Normas y referencias

4.1. Disposiciones legales y normas aplicadas

En este apartado se contemplará el conjunto de disposiciones legales (leyes, reglamentos, ordenanzas, etc.) y las normas de obligado cumplimiento que se han tenido en cuenta para la realización del TFG (Trabajo de Fin de Grado). La normativa a la que está sujeto este proyecto se puede dividir en dos partes principales:

1. Normativa de la Escuela Universitaria Politécnica para la realización de trabajos de final de grado:
    - Reglamento del trabajo de fin de grado para la Escuela Universitaria Politécnica de Ferrol.
    - UNE-EN-ISO 216. “Papel de escritura y ciertos tipos de impresos. Formatos acabados, series A y B”.
    - UNE 50132. “Numeración de las divisiones y subdivisiones en los documentos escritos”.
    - UNE 82100. “Magnitudes y unidades (partes 0 a 13)”.
2. Normativas que afectan a la calidad y exigencias de fabricación de los componentes utilizados para su comercialización y uso en la Unión Europea.
    - DIRECTIVA 2004/108/CE del Parlamento Europeo y del Consejo de 15 de diciembre de 2004 relativa a la aproximación de las legislaciones de los Estados miembros en materia de compatibilidad electromagnética.
    - UNE 20620-4:1980. “Materiales de base con recubrimiento metálico para circuitos impresos. Hoja de cobre”.

- UNE 21302-521:2004. “Vocabulario electrotécnico. Capítulo 521: Dispositivos semiconductores y circuitos integrados”.
- EN 123000/A1:1995. “Especificación genérica: Placas de circuitos impresos”. (Ratificada por AENOR en junio de 1996.)
- EN 123100:1992. “Especificación intermedia: placas de circuitos impresos de simple y doble cara con agujeros no metalizadas”. (Ratificada por AENOR en junio de 1996).

4.2. Bibliografía

[1] Charles L. Phillips, H. Troy Nagle, Jr. (1993) Sistemas de Control Digital. Análisis y diseño. Ed: Gustavo Gili, S.A.

[2] Katsuhiko Ogata. (1996). Sistemas de control en Tiempo discreto. Ed: Pearson Prentice Hall.

[3] Norman S. Nise (2002). Sistemas de control para ingeniería. Ed: Cecsa.

[4] Paul H. Lewis, Chang Yang (1999), Sistemas de control en ingeniería. Ed: Prentice Hall.

[5] John Dorsey (2005). Sistemas de control continuos y discretos. Ed: McGraw-Hill Interamericana.

4.3. Programas de cálculo

En esta sección se incluyen no sólo los programas de cálculo empleados para la realización del proyecto sino que tambien se mencionan los restantes programas empleados y que jugaron un papel muy importante en la realización del proyecto. De esta forma puede concluir que la lista de programas utilizados para la realización del presente proyecto ha sido:

- Mathworks Matlab r2013b.
- Arduino IDE 1.0.5.
- Processing 2.0.
- Proteus Professional v8.1.

4.4. Otras referencias

En este apartado se incluen otras referencias utilizadas para la realización del presente proyecto. No se trata de libros ya que estos se incluen en la sección de bibliografía. Se trata de otros documentos, partes de otros proyectos u referencias a páginas de Internet. A continuación se elabora la lista de referencias utilizadas:

- http://playground.arduino.cc/code/Keypad
- http://www.mathworks.es/hardware-support/arduino-matlab.html
- http://tallerarduino.com/category/videotutoriales/arduino-tutorials/
- http://tallerarduino.com/category/arduino/
- http://www.processing.org/

Capítulo 5

Definiciones y abreviaturas

Se incluyen en este apartado todas y cada una de las abreviaturas utilizadas a lo largo de la redacción de la presente memoria. Se trata de un apartado de consulta para facilitar la lectura y comprensión de la memoria.

- TFG: Trabajo de Fin de Grado.
- LTI: Linear Time-Invariant, Lineal e invariante con el tiempo
- DAQ: Data AcQuisition card, Tarjeta de adquisición de datos.
- A/D: Conversor de valores analógicos en digitales.
- D/A: Conversor de valores digitales en analógicos
- PWM: Pulse Width Modulation, modulación por ancho de pulso.
- LCD: Liquid Cristal Display, pantalla de cristal líquido.
- PID: Proporcional, integral y derivativo.
- EUP: Escuela Universitaria Politécnica.
- USB: Universal Serial Bus,
- FDT: Función De Transferencia.
- I2C: Inter-Integrated Circuit, Inter-Circuitos Integrados.

Capítulo 6

Requisitos de diseño

En este apartado se abarcará los requisitos de diseño impuestos de forma previa a la realización del proyecto. Los requisitos son de índoles diversas, es decir, se refieren al software, al hardware y a cuestiones de ergonomía. Éstos se exponen a continuación:

1. Se empleará el software y el hardware de Arduino para la realización del dispositivo emulador.

2. El emulador de funciones de transferencia debe brindar la posibilidad de simular el comportamiento de funciones de transferencia tanto continuas como discretas. En el caso de las funciones continuas se realizará mediante la aproximación más apropiada y con el periodo de muestreo más pequeño posible que permita el hardware de Arduino. En el caso de las funciones discretas el periodo de muestreo deberá ser seleccionable por el usuario teniendo también en cuenta las restricciones que impone el utilizar la plataforma Arduino. Las funciones de transferencia que se simularán serán del tipo SISO.

3. El dispositivo tendrá preinstaladas una serie de funciones de transferencia de sistemas tipo listas para simular.

4. El equipo debe funcionar de forma autónoma, es decir, no será necesario un ordenador para que funcione correctamente. Cabe mencionar que algunas prestaciones del equipo necesitarán de un ordenador para ser utilizadas. Por ejemplo, las prestaciones gráficas del equipo en las que se presentarán los datos de la simulación de forma gráfica exigen la utilización de un ordenador para ejecutar el software de presentación gráfica de datos. Sin embargo esto no impide que el equipo pueda ser utilizado sin un ordenador conectado a él.

5. El equipo debe contar con un LCD y un teclado con el que los usuarios puedan interactuar con él de una forma cómoda y ergonómica. Con estos dispositivos se pretende que el usuario interactúe con el emulador para, por ejemplo, seleccionar la función de transferencia que el usuario quiera simular en un determinado momento ó modificar el periodo de muestreo de la misma. Éstas y otras muchas opciones las podrá realizar el usuario mediante la interfaz implementada a través de estos dos dispositivos.

6. Una de las opciones del dispositivo emulador será la posibilidad de que el usuario introduzca nuevas funciones de transferencia tanto continuas como discretas de hasta orden 3.

7. Se debe incluir la opción de poder activar ó desactivar el envío de los datos de la simulación (entrada y salida) por el puerto serie para poder visualizarlos ó plotearlos en el ordenador al que se conecte el equipo.

8. El equipo deberá brindar la posibilidad de realizar la simulación de la función de transferencia de forma más real. Es decir, se podrá configurar, de forma previa a la simulación, una señal de ruido. Se deberán incluir cuatro opciones en cuanto a esto:

    - Sin ruido
    - Ruido del 0.005 % del valor teórico de la salida.
    - Ruido del 0.01 % del valor teórico de la salida.
    - Ruido del 0.1 % del valor teórico de la salida.

9. El periodo de muestreo deberá ser constante y el mínimo posible con el que se pueda garantizar una simulación correcta de las funciones de transferencia continuas. En las funciones de transferencia discretas será el usuario el que deba elegir el periodo de muestreo. Éste estará limitado en un rango entre 10 milisegundos y 1 segundo.

10. En la simulación de nuevas funciones de transferencia continuas el usuario podrá elegir entre dos tipos de aproximaciones: Euler y Tustin.

Cabe mencionar que uno de los requisitos ha sido la realización de dicho proyecto con el hardware y software de Arduino. Este dispositivo se trata de una plataforma de desarrollo open-source de bajo coste, basada en los microcontroladores Atmel. Esto hace implícitas al proyecto todas las ventajas (que son muchas) pero también todas las limitaciones que esto supone.

Capítulo 7

Análisis de las soluciones

Los requisitos de diseño impuestos previamente ya restringen bastante las soluciones a las que se pueden optar para la realización del presente proyecto. Por un lado, es requisito que la simulación de funciones de transferencia las haga un Arduino, es decir, ya no se pueden analizar soluciones con un hardware diferente. Sin embargo hubo que elegir entre los diferentes Arduinos disponibles: Uno, Mega, Leonardo, etc. Finalmente se obtó por el Arduino Mega ya que con el teclado y el LCD en un Arduino Uno ó Leonardo se ocuparían la mayoría de los pines de entrada/salida quedando las posibilidades de ampliación y mejora muy reducidas. Esto sucede con otros requisitos como son el teclado o el LCD donde sólo se optó por elegir algunas características específicas de los mismos como, por ejemplo, el display LCD es de cuatro líneas y veinte caracteres o el teclado matricial que es de 16 teclas.

A la hora de analizar la alimentación que llevaría el equipo se observa que la alimentación que proporciona el Arduino y que, es la obtenida del USB (Universal Serial Bus) del ordenador, es suficiente. Sin embargo, es requisito de diseño el que el equipo pueda funcionar de forma autónoma, es decir que no dependa de un ordenador para funcionar. Además como no todos los puertos USB de todos los ordenadores dan la misma corriente podría suceder que futuras modificaciones podrían hacer que la alimentación que proporciona el puerto USB de dicho ordenador no fuera suficiente para alimentar el Arduino y el display LCD. Teniendo esto en cuenta, se opta por diseñar una fuente de alimentación a medida para el equipo.

Para testear el equipo se necesitaría una tarjeta de adquisición de datos. Las tarjetas de adquisición de datos tienen un precio demasiado elevado además de que conllevan la necesidad de aprendizaje del software específico proporcionado por el fabricante ó de realizar una configuración específica para utilizarlas en un software

compatible. Por otro lado existen las soluciones libres en las que se incluyen tarjetas como Arduino las cuales tienen un precio muy inferior a las tarjetas de adquisición de datos comerciales. También se debe realizar una configuración específica para poder utilizarlas con software compatible y las limitaciones hardware son, obviamente, mayores. Sin embargo las prestaciones son suficientes para realizar la tarea de testeo del emulador.

Finalmente se utilizó un Arduino Leonardo y el software de Matlab a modo de tarjeta de adquisición de datos para el testeo del emulador de funciones de transferencia obteniendo unos resultados totalmente satisfactorios.

Capítulo 8

Resultados finales

Para abarcar el resultado final del presente proyecto, éste consta de la memoria, varios anexos, planos, estado de mediciones y presupuesto. En la sección actual se harán continuas referencias a estos documentos.

A continuación se explica de forma funcional y apoyada sobre un flujograma general el código fuente implementado y un ejemplo de utilización del equipo donde se pueden ver los resultados finales que se pueden esperar de él.

Antes de comenzar se recomienda ver los circuitos que se pueden encontrar en el documento Planos (sección IV). De esta forma se ven todas las conexiones que se han realizado en el Arduino (teclado, LCD, entrada, salida, acondicionamiento y puntos de prueba) y el código fuente será mucho más fácil de comprender. Existe un anexo dedicado exclusivamente al código fuente del equipo (sección 13) , sin embargo, aquí lo que se pretende es que el usuario entienda la secuencia de operaciones que realiza el equipo para obtener una simulación exitosa. Con lo cual se harán referencias continuas a este anexo para que pueda consultarse el código fuente.

8.1. Diagrama de flujo general

A continuación se muestra el diagrama de flujo general de uso del equipo. Las varialbles y funciones que se utilizan en cada proceso del diagrama serán explicadas en las secciones posteriores.

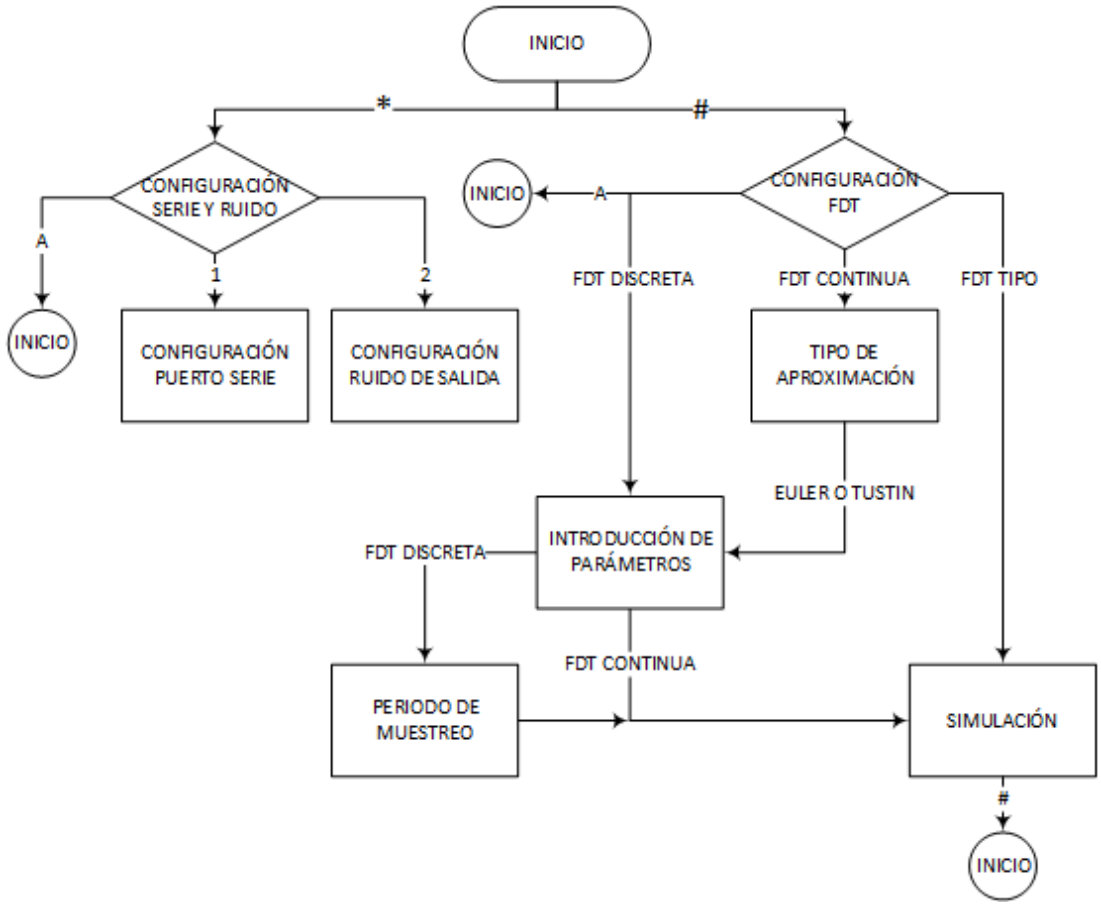


Figura 8.1.0.1 – Diagrama de flujo

8.2. Librerías y variables

En esta primera parte del código se incluyen las librerías necesarias para el funcionamiento de los periféricos y se definen las variables principales de todo el programa. Esta porción de código se corresponde con la sección 13.1 del anexo de código fuente del Arduino. En el hay notas aclaratorias de la función de cada variable. Sin embargo, para una mayor claridad, se exponen a continuación las funciones de las variables principales utilizadas en el código fuente:

- *Salida_Pwm*. No es una variable como tal, simplemente es para indicar que el pin por el que se va a tener la señal de salida en PWM será el pin 9 del Arduino Mega.

- *i*. En este caso es una variable global. En ella se almacena el instante de muestreo actual ya que en el programa se necesita constantemente saber el número de la muestra actual.

- *ruido*. Tendrá valores distintos según el ruido seleccionado por el usuario en el momento de la configuración del equipo.
- *x* e *y*. Son vectores de números flotantes donde se guardan los resultados de la simulación. Los valores de *x* serán las muestras tomadas de la señal de entrada y los valores de *y* serán los valores teóricos (sin ruido) que irá tomando la salida del sistema.
- *a3,a2,a1,a0*. Son los coeficientes del numerador de una función de transferencia determinada. Por ejemplo una función de transferencia donde el numerador sea $s^2 + 2s + 2$ tendrá los siguientes valores de esas variables: $a3 = 0; a2 = 1; a1 = 2; a0 = 2$.
- *b3,b2,b1,b0*. Lo mismo que los anteriores pero en este caso sobre los coeficientes del denominador.
- *k1,k2,k3,k4,k5,k6,k7*. Contienen los coeficientes calculados por el programa resultado de realizar la aproximación necesaria (Euler ó Tustin) en caso de las funciones de transferencia continuas ó de realizar la transformación directa a ecuaciones en diferencia de las funciones de transferencia discretas. Se puede encontrar más información sobre estos cálculos y transformaciones en el anexo cálculos (sección 10).
- *menu*. Es la variable de control del menú actual. Para cada pantalla distinta del lcd esta variable tendrá un determinado valor.
- *metodo*. False ó True en función de si el método de aproximación seleccionado es Tustin ó Euler respectivamente.
- *flag*. False ó True en función de si el envío de resultados de la simulación por el puerto serie esté desactivada ó activada respectivamente.
- *Aleatoriofloat*. Contiene el valor de la salida una vez aplicado el ruido que estuviera seleccionado en el momento anterior a la simulacion.

Con la lista de varibles anterior se puede entender el funcionamiento básico del programa y seguir el flujograma del mismo.

8.3. Configuración del LCD y del teclado

El trozo de código puede verse en la sección 2 del anexo de código fuente de Arduino. Éste contiene la configuración necesaria para el teclado y para el LCD. Hay que

añadir que previamente, en la sección anterior, ya se incluyen las librerías necesarias para que este código sea operativo.

En los comentarios adjuntos se explica brevemente la función del código escrito. Como puede observarse, el uso de las librerias de estos componentes facilita enormemente el trabajo.

Para el teclado tan sólo hay que indicar el número de filas, número de columnas, distribución de teclas y los pines del Arduino donde va a ser conectado. Se pueden verificar las conexiones en los circuitos de la sección IV. En la variable *key* siempre se tiene la última tecla pulsada.

En el caso del display LCD tan sólo se debe indicar el número de caracteres de cada línea y el número de líneas de que dispone el display LCD. En este caso también se indica la dirección I2C (Inter-Integrated Circuit, Inter-Circuitos Integrados) del módulo de comunicaciones I2C utilizado para la comunicación con el LCD. Este módulo de comunicaciones ha sido empleado para ahorrar cableado y pines del Arduino. Al utilizarlo tan sólo se deben utilizar dos pines de datos de Arduino y dos pines de alimentación. Los pines de datos irán conectados a los correspondientes pines del bus I2C del Arduino. Se pueden consultar los circuitos para mayor claridad (sección IV).

8.4. Configuración de la simulación

La configuración de la simulación se realiza en la función *setup* del Arduino. Para configurar la simulación, el usuario deberá interactuar por medio del teclado y del display LCD. Esta función se puede ver en el anexo de código fuente de Arduino (sección 13.2.

En primer lugar se realizan diversas configuraciones necesarias para el funcionamiento del equipo. Éstas se refieren a la semilla de los valores aleatorios para generar el ruido, la configuración de entradas y salidas, la selección de la frecuencia de la señal PWM de la salida así como la velocidad de comunicación de los datos del puerto serie. Estas son las líneas 1-20 de la función *setup* del Arduino (sección 13.2.

Las líneas de programación siguientes son las opciones de configuración de la simulación así como las de configuración de la función de transferencia que se desea

simular (líneas 20-290). Éstas han sido programadas en la rutina *setup* del Arduino. De esta forma se ve más claro el concepto de configuración del equipo para una posterior utilización del mismo (simulación) en la función *Loop* del Arduino. En primer lugar se ven las líneas de código para la configuración previa de la simulación (líneas 20-180). En ellas se puede ver que hacen uso de otras funciones. Éstas se pueden consultar en el anexo de Código Fuente del documento (sección 13).

La función *inicio_programa_lcd* tan sólo muestra la pantalla de bienvenida del dispositivo e insta al usuario a pulsar la tecla *[#]* para que pueda comenzar a configurar la función de transferencia deseada. Si por el contrario se pulsa la tecla *[*]* se entrará en las opciones de configuración del puerto serie y del ruido de la señal de salida. Esta función se puede consultar en la sección 13.4

Las líneas 20-180 abarcan la configuración del puerto serie y la configuración del ruido de la señal de salida. Como se puede ver es muy interactivo. Cabe destacar que se entra en estas opciones de configuración cuando el usuario pulsa la tecla *[*]* en la pantalla principal del programa. Como puede verse en el código, siempre está disponible la opción de pulsar la tecla *[A]* para volver Atrás, es decir, al menú anterior.

A continuación se verá la programación de la interfaz cuando el usuario pulse la tecla *[#]* estando el equipo en la misma pantalla principal (líneas 180-290). En estas líneas se verá la secuencia de pantallas a medida que el usuario vaya configurando la función de transferencia que desee emular.

En las primeras líneas (183-208) se observa el primer menú disponible cuando el usuario presione la tecla *[#]*. Este menú será un menú desplazable donde el usuario podrá elegir la función de transferencia deseada para emular. Se podrá regresar al menú principal, como siempre, pulsando la tecla *[A]*.

La función *Eleccion_func* a la que se le pasa el valor de la tecla pulsada es la encargada de realizar ese desplazamiento por el menú (sección 13.4. Para desplazarse se utilizan las teclas *[C]* y *[D]*. La primera se utiliza para desplazarse hacia arriba y la segunda hacia abajo.

Las siguientes líneas (208-245) se corresponde con la programación de la opción de función de transferencia continua elegida anteriormente por el ususario. Al tratarse de una función de transferencia continua tan sólo debe especificarse el método de

aproximación que se va a seguir y los parámetros del numerador y del denominador. La función *metodo_aprox* verifica la tecla pulsada por el usuario, *[1]* ó *[2]* si eligió el método de Euler ó el método de Tustin (sección 13.4). Una vez hecho esto se presenta por pantalla que el usuario introduzca el orden de la función de transferencia elegida: Pulsando las teclas *[1]*, *[2]* ó *[3]*. En la función *grado_continua* se guarda esta información introducida por el usuario (sección 13.4

En las líneas posteriores de código (245-259) se pedirán al usuario los parámetros principales del numerador y del denominador de la función de transferencia por medio de la funcion *parametros_grado* (sección 13.4). En las últimas líneas de esta captura del código ya se puede ver la primera línea que mostrará el display LCD mientras se esté simulando la función de transferencia.

En las últimas líneas (260-290) se programan las líneas que seguirá mostrando el display LCD mientras se esté simulando la función de transferencia. En éstas se indica si el puerto serie está activado ó no. La función *param_ec_dif* y la función *config_timer* tienen especial interés ya que es la forma en la que comienza la simulación. Por ello se verán a continuación para que se entienda mejor la secuencia de operaciones a realizar antes de comenzar la simulación de la función de transferencia.

8.5. Cálculo de parámetros

En este apartado se explica la función encargada de calcular los parámetros de la ecuación en diferencias resultante de una determinada función de transferencia (sección 13.4, líneas 722-805).

Cuando la variable *tipo* vale entre uno y tres (incluidos) la función de transferencia introducida por el usuario ha sido una función de transferencia continua introducida por él y ha elegido la aproximación de Euler para la simulación. El uno se corresponde con orden uno, dos con orden dos y tres con orden tres según lo hubiera seleccionado el usuario.

Cuando la variable *tipo* vale entre cuatro y seis (incluidos) el usuario ha seleccionado simular una función de transferencia continua con la aproximación de Tustin. El cuatro se corresponde con una función de transferencia de orden uno, el cinco con orden dos y el seis con orden tres.

Cuando la variable *tipo* vale entre siete y nueve (incluidos) el usuario ha seleccionado simular una función de transferencia discreta. Cuando *tipo* vale siete la función introducida ha sido de orden uno, ocho se corresponde con orden dos y nueve con orden tres.

El cálculo de los distintos parámetros *[k]* se puede consultar en el anexo cálculos (sección 10) donde se implementan las aproximaciones de Euler y de Tustin para funciones de transferencia de distinto orden: Desde orden 1 hasta orden 3.

8.6. Configuración del periodo de muestreo

A continuación se muestran las líneas de código de la función *config_timer*. En ella se realiza la configuración de la interrupción por desbordamiento del *timer1*. Éste será el encargado de que el periodo de muestreo sea el indicado y que sea siempre constante:

```
1  // Subrutina de configuración del timer 1
2  // Se encarga de que el periodo de muestreo sea el indicado.
3  // Se utiliza la interrupción por desbordamiento.
4  void config_timer()
5  {
6    cli();                      // Deshabilita interrupciones
7    TCCR1A = 0x00;
8    TCCR1B= 0x00;
9    TCCR1B = (1 << CS12);       //Preescaler de 256
10   TCNT1=(1.048576-T)/16e-6; // Tmax=1,04s; Tmin=16us (Teórico).
11   TIMSK1=(1<<TOIE1);          // Arranca el timer 1.
12   sei();                      // Habilita interrupciones
13 }
```

8.7. Simulación de la función de transferencia

La simulación de la función de transferencia se realiza siempre en la subrutina de interrupción por desbordamiento del timer 1. Éste se desborda, como se ha visto anteriormente, cada periodo de muestreo. El valor por defecto son 10 ms que a la vez es el valor mínimo que puede tomar y que garantiza un funcionamiento correcto. En la función *void loop* del Arduino tan sólo se llama al teclado para comprobar si el usuario ha pulsado la tecla *[#]*. En ese caso se pararía la simulación actual y se volvería al estado inicial del equipo.

```
1 //////////////////////////////////////////////////////////////////
2 // Programa principal
3 void loop()
4 {
5   key = keypad.getKey();      // Reinicio en caso de pulsar #.
6   if (key=='#') reinicio();
7 }
8 //////////////////////////////////////////////////////////////////
```

A continuacion se mostrará y se explicará de forma detallada el funcionamiento de la subrutina de vectorización de la interrupción por desbordamiento del *timer 1* del Arduino (sección 13.3). En esta subrutina se pueden diferenciar tres bloques funcionales:

1. Recarga del temporizador y señal de entrada.
2. Cálculo de la salida teórica según la función de transferencia y la señal de entrada leída. Cálculo de la salida con ruido.
3. Simulación de resultados. Escritura del valor de salida por el pin 9 del Arduino en forma de señal PWM. Envío de los datos de simulación por el puerto serie en caso de que éste esté activado.

El registro de conteo del temporizador se recarga con un valor dependiente del tiempo de muestreo necesario que hace que el temporizador se desborde siempre una vez transcurrido ese periodo de muestreo. La recarga de este registro se hace nada más ejecutarse la interrupción por desbordamiento para que el periodo sea siempre constante y ya comienze a contar una vez terminado el periodo anterior. Pueden verse señales de entrada de simulación que en este caso están comentadas y no forman parte del código pero que podrían ser útiles para pruebas. Entre las señales de simulación posibles se tiene un escalón del nivel indicado ó una señal senoidal aproximada. Estas señales de simulación han sido muy útiles a la hora de desarrollar y comprobar los resultados de las simulaciones.

En la segunda parte (líneas 22-46) se puede ver como se calcula la salida del sistema. Esta salida se calcula de forma diferente en función de si la función de transferencia es de grado 1, 2 ó 3. Aquí ya no se diferencia el método de aproximación ó el tipo de función de transferencia (continua ó discreta) puesto que esa diferencia ya ha quedado plasmada en los parámetros *k* que forman parte de la ecuación en diferencias resultante.

Se observa que se ha programado que para muestras anteriores a la cero el valor sea siempre nulo. Por ejemplo, cuando se comience a simular una función de orden 3, la primera muestra depende de las tres anteriores. En el programa se van almacenado las distintas muestras en un array, uno para la señal de entrada y otro para la señal de salida. De esta forma cuando se estuvieran simulando los tres primeros valores de la salida de la función de transferencia haría falta acceder al array de entrada ó al de salida con un índice negativo. Si se intenta, se accedería a las posiciones de memoria anteriores donde comienza el array. Lo que se conseguiría serían valores aleatorios que dependen de lo que hubiera en esas posiciones de memoria en ese instante. Para solucionar esto se ha programado que cuando se intente acceder a una muestra anterior al tiempo, es decir, al vector de entrada o salida con un índice negativo, el resultado sea siempre nulo.

Una vez calculada la salida teórica de la función de transferencia se llama a la funcion *Genera_ruido* para añadirle un ruido a la señal de salida en función de la opción que el usuario haya seleccionado. Como se mencionó anteriormente las opciones de ruido disponibles serían: Sin ruido, ruido del 0.005 %, ruido del 0.01 % y ruido del 0.1 %.

En la tercera y última parte (líneas 47-77) se puede ver la secuencia de simulación. Antes de comenzar a explicar el funcionamiento se debe explicar porqué sólo se tienen en cuenta cuatro valores en cada array, tanto el de la entrada como el de la salida. Al tratarse de funciones de transferencia de hasta orden 3, la ecuación en diferencias que se obtienen siempre van a depender de la entrada actual, las tres últimas muestras de la entrada y las tres últimas de la salida del sistema como máximo (orden 3). Esto hace que no se deban guardar todos los valores de entrada y salida de la simulación ejecutada ya que de esta forma lo que se conseguiría sería llenar la memoria del Arduino, pues hay que tener en cuenta que tanto la entrada como la salida se almacenan como números en coma flotante, es decir, cada uno ocupa 32 bytes en memoria. Lo que se ha implementado en este proyecto es un emulador de plantas de laboratorio, el sistema de adquisición y control sería el que tendría que hacer la función de grabar un histórico de valores de ser necesario.

De esta forma, se diferencian dos partes principales, una para las primeras cuatro muestras (contando la muestra cero) y otra para las muestras restantes que podrán ser infinitas. En ambos bloques se verifica si el puerto serie está activado y, si es así, se envía la información por el puerto serie. En los dos bloques se escribe en la sa-

lida PWM el valor de salida obtenido haciendo el escalado correspondiente a la salida PWM, es decir, para el valor maximo de salida (100 %) el valor del “duty cicle”de la señal PWM será máximo (255 en Arduino), y para el valor mínimo (0 %) el “duty cicle”será mínimo (0 en Arduino).

La diferencia entre los dos bloques es que en el primero se aumenta el índice del vector de salida y de entrada (primeras cuatro muestras) pero en el segundo no se hace esto. Lo que se hace aquí es rotar a izquierdas tanto el vector de muestras de la entrada como el vector de muestras de la salida. Haciendo esto no destruímos datos que hagan falta ya que en la muestra 4 sólo se necesitan las tres muestras anteriores (entrada y salida) y la actual en caso de la entrada, en la muestra 5 igual, etc.

8.8. Funciones de transferencia tipo

El dispositivo está equipado con una serie de funciones de transferencia tipo. Todas estas funciones de transferencia son continuas y con unos parámetros precargados:

1. Motor de CC: En este caso se trata de una función de transferencia tipo. Se corresponde con un determinado modelo real de un motor de corriente continua. Se relaciona la velocidad del motor con la tensión aplicada. La función de transferencia es:

$$G(s) = \frac{0{,}01}{0{,}005s^2 + 0{,}06s + 0{,}1001} \tag{8.8.0.1}$$

2. Circuito RLC: Se corresponde con un circuito real formado por una resistencia, una bobina y un condensador. Se relaciona la tensión en el condensador con la tensión de entrada al circuito. La función de transferencia es:

$$G(s) = \frac{17}{s^2 + 8s + 17} \tag{8.8.0.2}$$

3. Masa oscilante: Se corresponde con una función de tranferencia formada por una masa, un muelle y un amortiguador viscoso. Se relaciona el desplazamiento de la masa con la fuerza aplicada. La función de transferencia es:

$$G(s) = \frac{0{,}0192}{s^2 + 2{,}353s + 3{,}84} \tag{8.8.0.3}$$

4. Control de temperatura: Se trata de una función de transferencia correspondiente al control de la temperatura de un local por medio de un radiador eléctrico. Se relaciona la tensperatura del local con la temperatura de referencia. La función de transferencia es:

$$G(s) = \frac{0{,}001}{s^2 + 0{,}041s + 0{,}0012} \tag{8.8.0.4}$$

5. Tanques acoplados: Se trata de una función de transferencia correspondiente a dos tanques acoplados. Se relaciona la altura del último tanque con el caudal de entrada al primero. La función de transferencia es:

$$G(s) = \frac{1{,}5}{1800s^2 + 85s + 1} \tag{8.8.0.5}$$

6. Aeropéndulo. Se corresponde con la función de transferencia de un aeropéndulo: Un péndulo con un ventilador en la parte inferior que hace inclinarse al péndulo al girar a alta velocidad. La funcion de transferencia es:

$$G(s) = \frac{0{,}001}{s^2 + 0{,}041s + 0{,}0012} \tag{8.8.0.6}$$

7. AmortiguadorCarroResorte: Sistema compuesto por un amortiguador, un carro y un resorte. Se relaciona el desplazamiento final con la fuerza aplicada. La función de transferencia es:

$$G(s) = \frac{1}{s^3 + 4s^2 + 5s + 2} \tag{8.8.0.7}$$

8. Nivel de un depósito. Se trata de la función de transferencia del nivel de un depósito. Relaciona el nivel del depósito con el caudal de entrada. La función de transferencia real es la siguiente:

$$G(s) = \frac{1{,}5}{40s+1} \cdot e^{-0{,}2s} \tag{8.8.0.8}$$

Sin embargo, la función de transferencia que se va a emular en el equipo es distinta. El retardo de la función de transferencia anterior ha sido aproximado por medio de la función *pade* de Matlab (orden 2). Esta función ayuda a cambiar las funciones de transferencia con retardos a funciones de transferencia sin ellos. La función de transferencia resultante ha sido:

$$G(s) = \frac{1{,}5s^2 - 45s + 450}{40s^3 + 1201s^2 + 12030s + 300} \tag{8.8.0.9}$$

9. Control de temperatura de un proceso: Se trata de controlar la temperatura de un líquido por medio de un radiador eléctrico. Relaciona la temperatura del líquido con la potencia entregada al radiador. La función de transferencia real es:

$$G(s) = \frac{5000}{s^2 + 60s + 500} \cdot e^{-0{,}1s} \tag{8.8.0.10}$$

Al igual que la anterior, el retardo de la función de transferencia ha sido aproximado por medio de la función pade de Matlab (orden 1). La función de transferencia resultante ha sido:

$$G(s) = \frac{-5000s + 100000}{s^3 + 80s^2 + 1700s + 10000} \tag{8.8.0.11}$$

10. Sistema de mezclado de dos productos de diferentes densidades con retardo de transporte de fluido desde la válvula hasta el tanque de mezcla. Relaciona la densidad del fluido resultante con el producto de la densidad y el caudal de entrada de uno de los fluidos. La función de transferencia real es:

$$G(s) = \frac{1}{15s+2} \cdot e^{-1{,}4s} \tag{8.8.0.12}$$

Al igual que la anterior, el retardo de la función de transferencia ha sido aproximado por medio de la función pade de Matlab (orden 2). La función de transferencia resultante ha sido:

$$G(s) = \frac{s^2 - 4{,}286s + 6{,}122}{15s^3 + 66{,}29s^2 + 100{,}4s + 12{,}24} \tag{8.8.0.13}$$

11. Posicionamiento de antena parabólica. Relaciona la posición angular de la antena con la tensión de entrada al motor que la mueve. La función de transferencia del sistemas es:

$$G(s) = \frac{1}{2s^3 + 3s^2 + 2s} \tag{8.8.0.14}$$

12. Posicionamiento de cámaras de escenario. Relaciona la posición angular de dos cámaras con la tensión de alimentación del motor que las posiciona. La función de transferencia del sistema es:

$$G(s) = \frac{123632{,}8}{s^3 + 87{,}6s^2 + 5329s} \tag{8.8.0.15}$$

13. Posicionamiento angular de una carga. Relaciona el posicionamiento angular de una carga con un posicionamiento angular de referencia. La función de transferencia del sistema es:

$$G(s) = \frac{12968{,}469}{s^3 + 666{,}911s^2 + 288{,}315s} \tag{8.8.0.16}$$

14. Función de movimiento de un barco. La función de transferencia del sistema es:

$$G(s) = \frac{0{,}0000828157}{s^2 + 0{,}00472s} \tag{8.8.0.17}$$

Además el equipo permite introducir nuevas funciones de transferencia de distintos tipos:

1. Función discreta: Permite al usuario introducir una función de transferencia en tiempo discreto de hasta orden 3. El orden de la función de transferencia y los parámetros del numerador y del denominador de dicha función se pedirán en un menú posterior. También se pedirá el periodo de muestreo de la misma (entre 10ms y 1s).

2. Función continua: Permite al usuario introducir una función de transferencia en tiempo continuo de hasta orden 3. El método de aproximación de la misma, el orden de la función de transferencia y los parámetros del numerador y del denominador de dicha función se pedirán en un menú posterior. En este caso no tiene sentido hablar de periodo de muestreo ya que el equipo utilizará el mínimo posible: 10ms.

8.9. Ejemplo de utilización

En el presente apartado se realiza un ejemplo básico de control PID realizado mediante un Arduino Leonardo empleando el software de Matlab sobre una función de transferencia emulada mediante el equipo desarrollado. Para una correcta conexión del equipo se debe consultar el documento planos (sección IV) donde el plano número 2 refleja el montaje realizado en esta ocasión para probar el funcionamiento del equipo.

Además de estas pruebas, se han realizado unas pruebas más simples, que se muestran en el anexo 11, donde lo que se comprobó fue que el periodo de muestreo permanece constante, y que la respuesta del sistema emulado coincide con la simulación en Matlab del la función de transferencia correspondiente.

Para poder usar Arduino con Matlab r2013b debe instalarse antes. La instalación de Arduino en Matlab puede resumirse en los siguientes pasos:

1. Ejecutar el comando *targetinstaller* en la ventana principal de Matlab

2. Seguir los pasos del asistente de instalación de Matlab y elegir la opción de Arduino cuando proceda, esto descargará de Internet los paquetes necesarios de Arduino para que funcione correctamente en Matlab.

3. Una vez descargados se debe añadir la carpeta donde se descargaron al Path de Matlab. La carpeta por defecto es:

C:/MATLAB/SupportPackages/R2013b/downloads/arduinodownload.

4. Ejecutar el comando *install_arduino* en la pantalla principal de Matlab.

5. Cargar el programa *adiorv.pde* en el Arduino (mediante su propio software) en el Arduino con el que se va a trabajar con Matlab.

6. Usar el Arduino en Matlab comenzando con la función $x = arduino('COMXX')$ donde la primera *x* es el nombre que se quiera y las dos siguientes son el número del puerto *COM* que el ordenador le ha asignado al Arduino.

El programa de Matlab utilizado para el control PID mediante el Arduino Leonardo consta de cuatro módulos: El módulo principal, módulo de configuración del sistema, módulo de configuración del regulador y el módulo de visualización de los datos. Estos cuatro módulos se pueden encontrar en el anexo de código fuente PID Matlab (sección 15). No se entra a explicar el PID implementado con Matlab ya que no es el objetivo, sino que lo importante es comprobar que se puede realizar un control sobre la planta virtual.

Se debe ejecutar el módulo principal (sección 15.1) para que todo funcione correctamente ya que éste se encarga de llamar a los demás.

En el módulo de configuración del sistema (sección 15.2) se puede modificar el tiempo de muestreo con el que funcionará el regulador. Se recomienda que éste no sea inferior a 100 ms teniendo el envío de datos por el puerto serie del equipo emulador desactivado. Si está activado se aconseja que el periodo de muestreo no sea inferior a 300 ms. Todos estos valores dependen del ordenador en el que vaya a ser utilizado y de las aplicaciones y procesos que se estén ejecutando en ese momento en el ordenador.

En el módulo de configuración del PID (sección 15.3) se pueden cambiar los parámetros principales del regulador PID.

Por último tenemos el módulo de visualización de los datos (sección 15.4). En este caso se emplea una visualización de datos en tiempo real. Sin embargo al final de la prueba se presentarán la totalidad de los datos en una gráfica.

Lo que se debe hacer para ejecutar la prueba es ejecutar primeramente el programa *MyMain* (sección 15.1) desde Matlab y este comenzará a mostrar la gráfica

de respuesta en unos segundos. Se distinguen tres líneas principales: La consigna, la señal de control (la entrada al emulador) y la variable de proceso (la salida del emulador). Si todo es correcto, la señal de salida debe ser nula y la señal de entrada proveniente del PID debe estar al 100 % .

Ahora es el momento de simular la función de transferencia que se desee controlar. Al principio de cada gráfica se verá una zona donde la señal de control proveniente del PID está siempre al 100 %. Esta zona es donde todavía no se ha comenzado a simular la función de transferencia, es decir, donde la salida del sistema es todavía nula. Las gráficas 8.9.0.2, 8.9.0.3 y 8.9.0.4 mostrarán los resultados de simular distintas funciones de transferencia. En ellas la línea roja será la señal de control proveniente del Arduino Leonardo por medio de Matlab, la azul será la consigna y la verde la salida del sistema capturada con el Arduino Leonardo. Puede ocurrir que el regulador PID no esté perfectamente ajustado ya que no es el propósito del presente proyecto.

Las funciones de transferencia que se van a controlar a modo de ejemplos de funcionamiento son las siguientes:

1. Función de transferencia de orden 1.

$$G(s) = \frac{1}{s+1} \tag{8.9.0.18}$$

2. Control de temperatura (incluida en las funciones tipo del equipo).

$$G(s) = \frac{0{,}001}{s^2 + 0{,}041s + 0{,}0012} \tag{8.9.0.19}$$

3. Nivel de un depósito (incluida en las funciones tipo del equipo).

$$G(s) = \frac{1{,}5s^2 - 45s + 450}{40s^3 + 1201s^2 + 12030s + 300} \tag{8.9.0.20}$$

1. Control PID sobre la función de transferencia de la ecuación 8.9.0.18

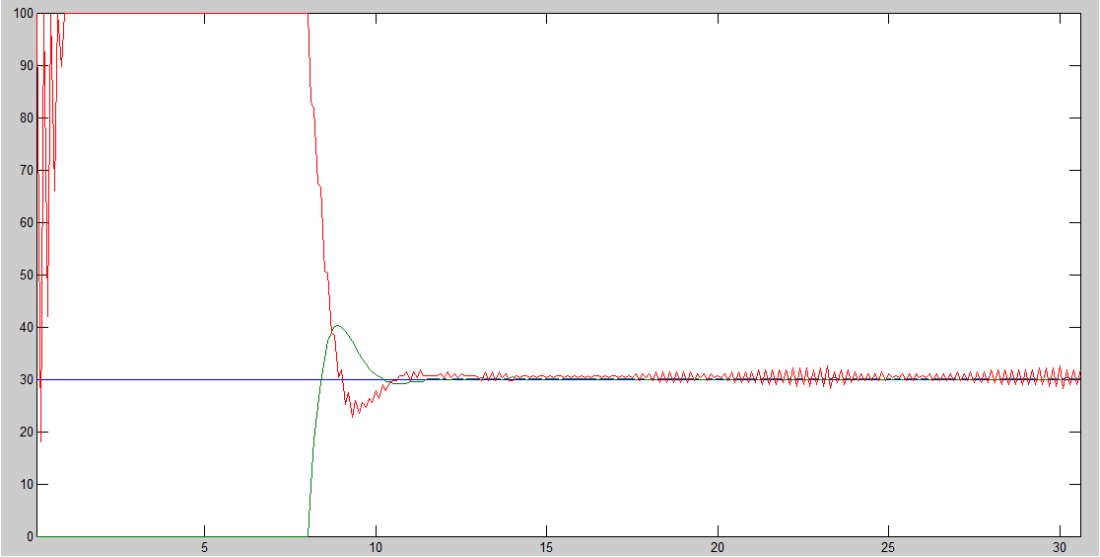


Figura 8.9.0.2 – Control PID 1

2. Control PID sobre la función de transferencia de la ecuación 8.9.0.19

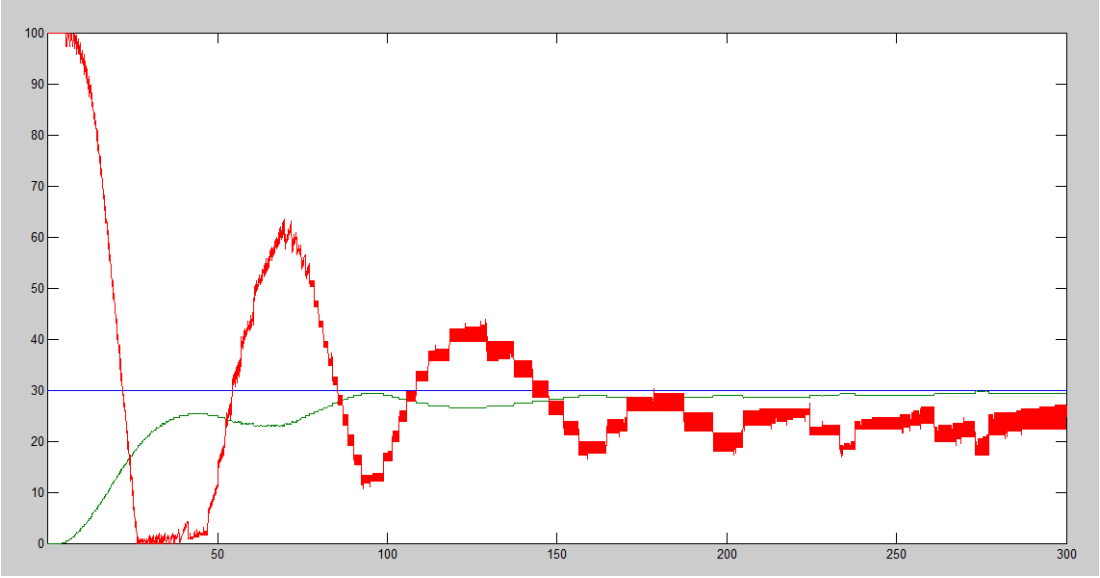


Figura 8.9.0.3 – Control PID 2

3. Control PID sobre la función de transferencia de la ecuación 8.9.0.20

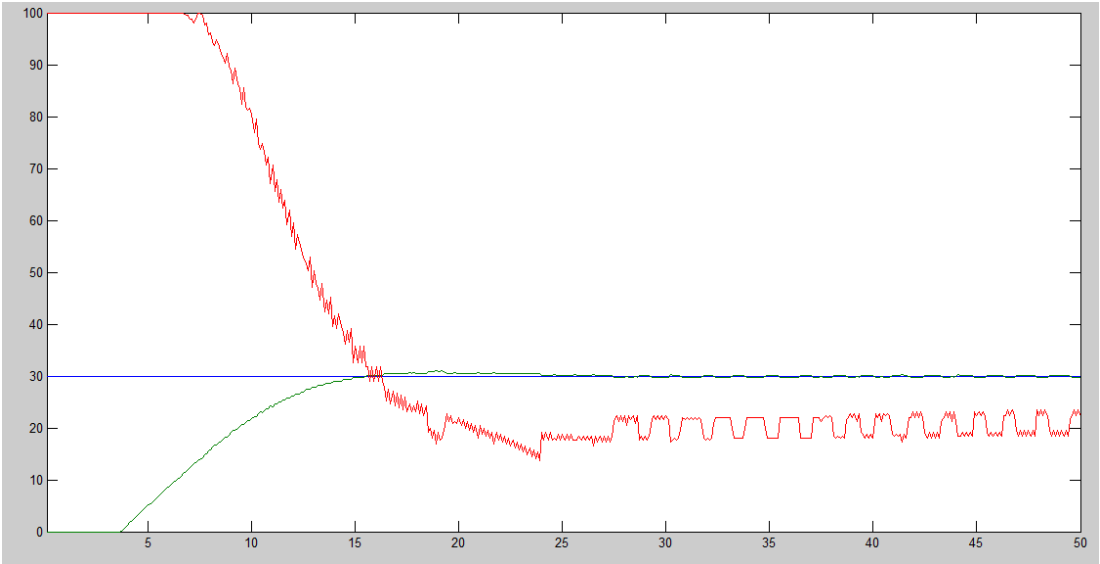


Figura 8.9.0.4 – Control PID 3

Como se puede observar el equipo se comporta de una forma muy similar a una planta real. Existen muchas combinaciones que se podrían probar con el equipo pues las posibilidades que brinda son muchas. Pero con estas pruebas es suficiente para demostrar la funcionalidad del equipo ya que el testeo de resultados ha sido satisfactorio.

Las pruebas anteriores han sido realizadas con funciones de transferencia teóricas sin ruido alguno a su salida. A continuación se va a realizar el mismo control PID pero esta vez se emplearán las distintas opciones que ofrece el dispositivo en cuanto al ruido de la señal de salida. Para ello se emplea la función de transferencia de la ecuación 8.9.0.20. Los resultados obtenidos han sido los siguientes:

1. Ruido del 0.005 %.

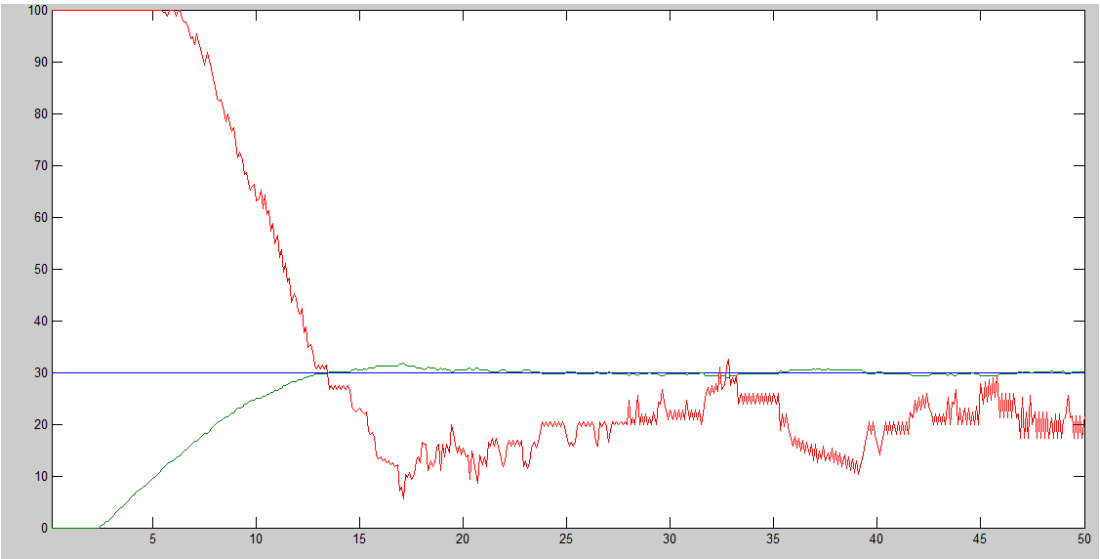


Figura 8.9.0.5 – Control PID ruido 1

2. Ruido del 0.01 %.

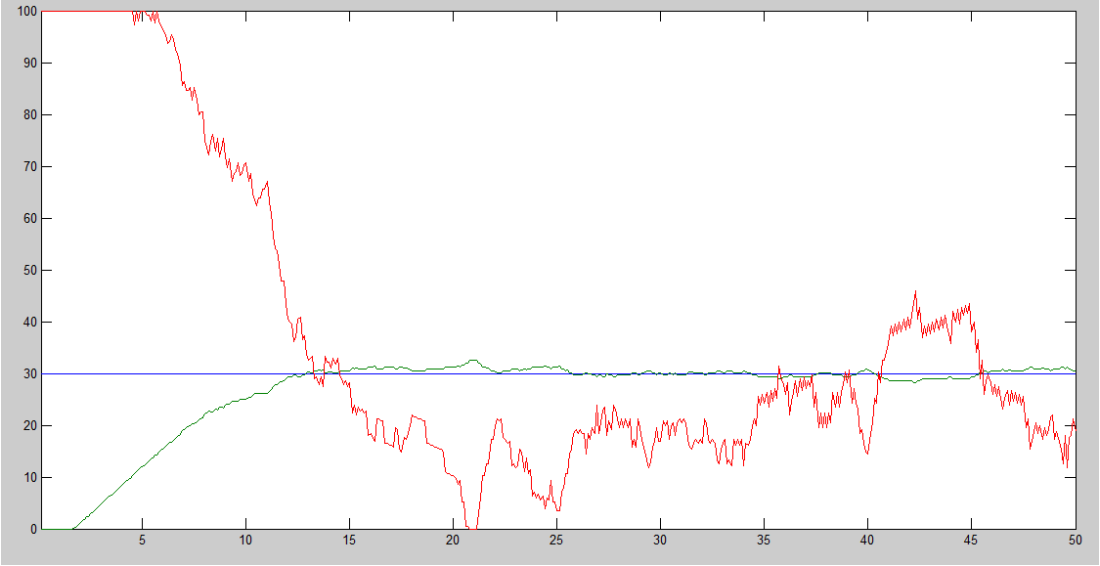


Figura 8.9.0.6 – Control PID ruido 2

3. Ruido del 0.1

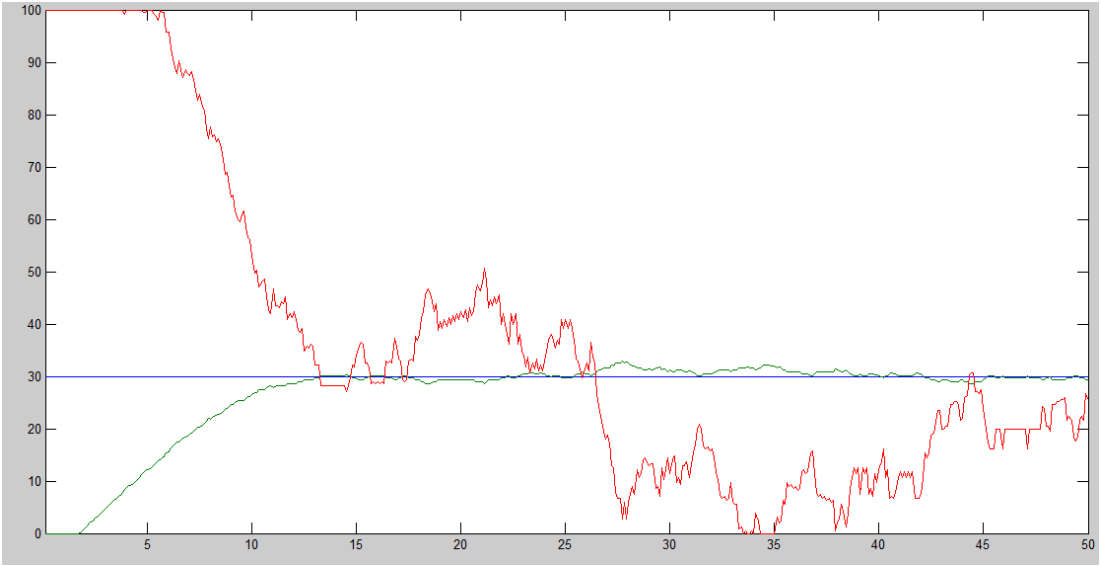


Figura 8.9.0.7 – Control PID ruido 3

En las figuras 8.9.0.5, 8.9.0.6 y 8.9.0.7 se puede ver la importancia del ruido. Aunque el porcentaje de ruido introducido es muy bajo se observa que cobra una gran importancia. Esto se debe a que el ruido se aplica a una señal de salida en la que la muestra previa ya tiene un ruido introducido. El ruido se manisfiesta sobre todo en la variabilidad de la señal de control proveniente del regulador PID. Se concluye entonces que la característica del ruido de la salida funciona correctamente al igual que el resto del equipo.

ANEXOS

TÍTULO: **DESARROLLO Y TESTEO DE UN EMULADOR DE PLANTAS INDUSTRIALES BASADO EN ARDUINO**

ANEXOS

PETICIONARIO: **ESCOLA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

15405 - FERROL

FECHA: JUNIO DE 2014

AUTOR: ESTEBAN VELO SÁNCHEZ

Fdo.:

Capítulo 9

Anexo 1: Documentación de partida

En este anexo se adjunta el justificante de asignación del trabajo de fin grado donde se recoge el título del trabajo de fin de grado, sus tutores así como el número de TFG asignado.

ESCUELA UNIVERSITARIA POLITÉCNICA

ASIGNACIÓN DE TRABAJO FIN DE GRADO

En virtud de la solicitud efectuada por:
En virtud da solicitude efectuada por:

APELLIDOS, NOMBRE: *Velo Sánchez,Esteban*
APELIDOS E NOME:

DNI: **Fecha de Solicitud:** FEB2014
DNI: *Fecha de Solicitude:*

Alumno de esta escuela en la titulación de Grado en Ingeniería en Electrónica Industrial y Automática, se le comunica que la Comisión de Proyectos ha decidido asignarle el siguiente Trabajo Fin de Grado:

O alumno de esta escola na titulación de Grado en Enxeñería en Electrónica Industrial e Automática, comunícaselle que a Comisión de Proxectos ha decidido asignarlle o seguinte Traballo Fin de Grado:

Título T.F.G:Desarrollo y testeo de un emulador de plantas industriales basado en Arduino

Número TFG: *770G01A52*

TUTOR:(Titor) *Calvo Rolle,Jose Luis*

COTUTOR/CODIRECTOR: *José Luis Casteleiro Roca*

La descripción y objetivos del Trabajo son los que figuran en el reverso de este documento*:*

A descrición e obxectivos do proxecto son os que figuran no reverso deste documento.

Ferrol a Lunes, 2 de Junio del 2014

Retirei o meu Traballo Fin de Grado o dia _____ de _____ do ano _____

Fdo: Velo Sánchez,Esteban

DESCRIPCIÓN Y OBJETIVO:Objeto:
El objetivo del proyecto es el desarrollo, cálculo, diseño, implementación y testeo de un emulador de funciones de transferencia discretas. El objetivo del montaje es conseguir un dispositivo autónomo, cuya finalidad es la realización de prácticas de laboratorio basadas en tarjetas de adquisición de datos, sin tener que disponer de una planta real. El emulador debería de poder permitir la variación tanto de los parámetros de la función de transferencia, como del periodo de muestreo de la misma.

Alcance:
- Aprendizaje de arduino sobre Matlab como tarjeta de adquisición de datos.
- Comprobación del funcionamiento sobre planta de laboratorio.
- Diseño de un circuito basado en arduino para emular funciones de transferencia discretas SISO, que cumpla las siguientes características:
  - o Interfaz LCD y teclado para poder interactuar.
  - o Incluir una serie de funciones de transferencia tipo.
  - o Posibilidad de introducir nuevas funciones de transferencia.
- Testeo y comprobación del correcto funcionamiento del circuito empleando Arduino como DAQ.

Capítulo 10

Anexo 2: Cálculos

Este anexo está dedicado a la exposición de los cálculos necesarios para la realización del presente proyecto. Se pueden dividir en cuatro bloques fundamentales.

1. Aproximaciones de funciones de transferencia en tiempo continuo a tiempo discreto.: La aproximación de Euler y la aproximación de Tustin.
2. Transformaciones a ecuaciones en diferencias de funciones de transferencia en tiempo discreto.
3. Cálculos de la placa de acondicionamiento.
4. Cálculos de la fuente de alimentación.

10.1. Las aproximaciones de Euler y de Tustin

Estas dos aproximaciones son muy diferentes en cuanto a los cálculos que deben realizarse. En la teoría de control la que más se ajusta a la realidad es la aproximación de Tustin. En la aproximación de Euler un sistema inestable u oscilante puede convertirse en estable y viceversa, sin embargo, en la aproximación de Tustin esto no sucede. El motivo de incluir ambas aproximaciones es simplemente didáctico. De esta forma se podrán ver las diferencias entre una función de transferencia aproximada por Euler ó por Tustin. Ambas aproximaciones han sido realizadas para funciones de transferencia de primer, segundo y tercer orden. Cabe destacar que bastaría con realizar los cálculos para la transformación de la función de transferencia con mayor grado que se vaya a manejar puesto que, en caso de querer simular una función de transferencia de menor grado bastaría con tener los términos de mayor grado nulos. Sin embargo se realizaron los cálculos para los distintos grados por claridad del código.

Este tipo de transformaciones consisten en realizar una asignación exacta del plano Z al plano S. A continuación se detallará cada una de ellas y finalmente se habla de las diferencias fundamentales entre ambas.

10.1.1. La aproximación de Euler

Consiste en sustituir en una función de transferencia en tiempo continuo, la variable *s* por la transformación mostrada en 10.1.1.1, donde T es el periodo de muestreo:

$$s = \frac{z-1}{Tz} \tag{10.1.1.1}$$

1. Funciones de transferencia de orden 1.
   Se supone una función de transferencia de orden uno con los siguientes coeficientes:
   $$G(s) = \frac{a_1 s + a_0}{b_1 s + b_0} \tag{10.1.1.2}$$
   Sustituyendo la *s* por la transformación de Euler indicada en 10.1.1.1 se otiene la función de transferencia en z:
   $$G(z) = \frac{a_1 \frac{z-1}{Tz} + a_0}{b_1 \frac{z-1}{Tz} + b_0} = \frac{a_1(z-1) + a_0 Tz}{b_1(z-1) + b_0 Tz} = \frac{(a_1 + a_0 T)z - a_1}{(b_1 + b_0 T)z - b_1} \tag{10.1.1.3}$$

2. Funciones de transferencia de orden 2. Se supone una función de transferencia de orden dos con los siguientes coeficientes:
   $$G(s) = \frac{a_2 s^2 + a_1 s + a_0}{b_2 s^2 + b_1 s + b_0} \tag{10.1.1.4}$$
   Sustituyendo la *s* por la transformación de Euler indicada en 10.1.1.1 se obtiene:
   $$G(z) = \frac{a_2(\frac{z-1}{Tz})^2 + a_1 \frac{z-1}{Tz} + a_0}{b_2(\frac{z-1}{Tz})^2 + b_1 \frac{z-1}{Tz} + b_0} = \frac{(a_2 + a1T + a_0 T^2)z^2 - (2a_2 + a_1 T)z + a_2}{(b_2 + b1T + b_0 T^2)z^2 - (2b_2 + b_1 T)z + b_2} \tag{10.1.1.5}$$

3. Funciones de transferencia de orden 3. Se supone una función de transferencia de orden tres con los siguientes coeficientes:
   $$G(s) = \frac{a_3 s^3 + a_2 s^2 + a_1 s + a_0}{b_3 s^3 + b_2 s^2 + b_1 s + b_0} \tag{10.1.1.6}$$
   Sustituyendo la *s* por la transformación de Euler indicada en 10.1.1.1 se obtiene:
   $$G(z) = \frac{a_3(\frac{(z-1)}{Tz})^3 + a_2(\frac{z-1}{Tz})^2 + a_1 \frac{z-1}{Tz} + a_0}{b_3(\frac{(z-1)}{Tz})^3 + b_2(\frac{z-1}{Tz})^2 + b_1 \frac{z-1}{Tz} + b_0} \tag{10.1.1.7}$$
   $$G(z) = \frac{(a_3 + a_2 T + a_1 T^2 + a_0 T^3)z^3 - (3a_3 + 2a_2 T + a_1 T^2)z^2 + (3a_3 + a_2 T)z - a_3}{(b_3 + b_2 T + b_1 T^2 + b_0 T^3)z^3 - (3b_3 + 2b_2 T + b_1 T^2)z^2 + (3b_3 + b_2 T)z - b_3} \tag{10.1.1.8}$$

Esta transformación se basa en datos del pasado para predecir el futuro. Es decir, se calcula la pendiente de la tangente a la función en cada punto y se avanza según ésta hasta el siguiente punto (en este caso el siguiente periodo de muestreo). La línea quebrada que se va formando de esta forma se denomina poligonal de Euler.

10.1.2. La aproximación de Tustin

La aproximación de Tustin (también conocida con el nombre de transformación bilineal) es usada habitualmente en el campo del procesamiento digital de señales y en la Teoría de control de señales discretas. Consiste en sustituir en una función de transferencia en tiempo continuo, la variable s por la siguiente transformación, donde T es el periodo de muestreo:

$$s = \frac{2}{T} \cdot \frac{z-1}{z+1} \tag{10.1.2.1}$$

1. Funciones de transferencia de orden 1.
   Sustituyendo la s por la transformación de Tustin indicada en 10.1.2.1 se obtiene la función de transferencia en z:

$$G(z) = \frac{\frac{2a_1(z-1)}{T(z+1)} + a_0}{\frac{2b_1(z-1)}{T(z+1)} + b_0} = \frac{(2a_1 + a_0T)z - 2a_1 + a_0T}{(2b_1 + b_0T)z - 2b_1 + b_0T} \tag{10.1.2.2}$$

2. Funciones de transferencia de orden 2. Sustituyendo la s por la transformación de Tustin indicada en 10.1.2.1 se obtiene:

$$G(z) = \frac{\frac{4a_2(z-1)^2}{T^2(z+1)^2} + \frac{2a_1(z-1)}{T(z+1)} + a_0}{\frac{4b_2(z-1)^2}{T^2(z+1)^2} + \frac{2b_1(z-1)}{T(z+1)} + b_0} \tag{10.1.2.3}$$

$$G(z) = \frac{(4a_2 + 2a_1T + a_0T^2)z^2 + (2a_0T^2 - 8a_2)z + 4a_2 + a_0T^2 - 2a_1T}{(4b_2 + 2b_1T + b_0T^2)z^2 + (2b_0T^2 - 8b_2)z + 4b_2 + b_0T^2 - 2b_1T} \tag{10.1.2.4}$$

3. Funciones de transferencia de orden 3. Sustituyendo la s por la transformación de Tustin indicada en 10.1.2.1 se obtiene:

$$G(z) = \frac{a_3(\frac{(z-1)}{Tz})^3 + a_2(\frac{z-1}{Tz})^2 + a_1\frac{z-1}{Tz} + a_0}{b_3(\frac{(z-1)}{Tz})^3 + b_2(\frac{z-1}{Tz})^2 + b_1\frac{z-1}{Tz} + b_0} \tag{10.1.2.5}$$

$$G(z) = \frac{(8a_3 + 4a_2T + 2a_1T^2 + a_0T^3)z^3 + (-24a_3 - 4a_2T + 2a_1T^2 + 3a_0T^3)z^2}{(8b_3 + 4b_2T + 2b_1T^2 + b_0T^3)z^3 + (-24b_3 - 4b_2T + 2b_1T^2 + 3b_0T^3)z^2} \tag{10.1.2.6}$$

$$\frac{(24a_3 - 4a_2T - 2a_1T^2 + 3a_0T^3)z + (-8a_3 + 4a_2T - 2a_1T^2 + a_0T^3)}{(24b_3 - 4b_2T - 2b_1T^2 + 3b_0T^3)z + (-8b_3 + 4b_2T - 2b_1T^2 + b_0T^3)} \tag{10.1.2.7}$$

   La solución son las dos ecuaciones anteriores juntas, es decir, la segunda ecuación tan sólo es la continuación del la división anterior.

Este método utiliza la integración trapezoidal para aproximar la función de transferencia. De ahí que sea más preciso que el anterior.

10.1.3. Diferencias entre las aproximaciones

Las diferencias fundamentales que se pueden observar entre ambas son la región de estabilidad y la dinámica de la respuesta. La aproximación de Tustin conserva la estabilidad de la función de transferencia original, es decir, si esta es estable la aproximada también lo será y viceversa. En la de Euler no necesariamente ocurrirá esto y será inestable para funciones de transferencia con polos muy rápidos. Las regiones de estabilidad para ambas pueden verse en la figura 10.1.3.1:

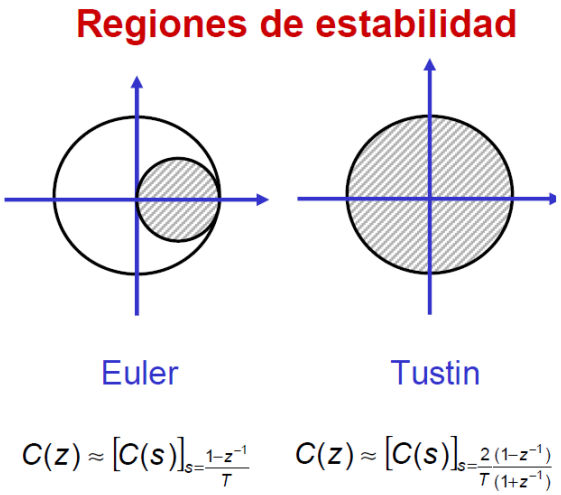


Figura 10.1.3.1 – Regiones de estabilidad

En cuanto a la dinámica la aproximación de Tustin es más precisa puesto que usa integración trapezoidal en la transformación.

10.2. Transformaciones a ecuaciones en diferencia

Una vez obtenida la función de transferencia discreta, ésta debe convertirse a ecuaciones en diferencias para poder ser manejada por el Arduino. Este paso es muy sencillo. Tan sólo hacen falta los parámetros anteriormente calculados y la transformada Z inversa de la ecuación 10.2.0.1 (donde i es el instante de muestreo actual y k un número natural que hace referencia a un estado anterior):

$$z^{-k}X(z) \Rightarrow X(i-k) \tag{10.2.0.1}$$

A continuación se va a realizar un caso general de transformación a ecuaciones en diferencias de una función de transferencia de orden 2 en Z:

$$G(z) = \frac{Y(z)}{X(z)} = \frac{a_2z^2 + a_1z + a_0}{b_2z^2 + b_1z + b_0} = \frac{a_2 + a_1z^{-1} + a_0z^{-2}}{b_2 + b_1z^{-1} + b_0z^{-2}} \tag{10.2.0.2}$$

Multiplicando en cruz y operando se obtiene lo siguiente:

$$b_2 y(z) + b_1 z^{-1} y(z) + b_0 z^{-2} y(z) = a_2 x(z) + a_1 z^{-1} x(z) + a_0 z^{-2} x(z) \qquad (10.2.0.3)$$

Aplicando la antitransformada de la ecuación 10.2.0.1 y despejando y(k) se obtiene:

$$y(k) = \frac{a_2}{b_2} x(k) + \frac{a_1}{b_2} x(k-1) + \frac{a_0}{b_2} x(k-2) - \frac{b_1}{b_2} y(k-1) - \frac{b_0}{b_2} y(k-2) \qquad (10.2.0.4)$$

De esta se ha calculado la ecuación en diferencias de una función de transferencia en *z* de orden 2 genérica. En el código fuente los coeficientes que acompañan a cada término *x(k)*, *x(k-1)*...hasta *x(k-n)*, *y(k-1)* hasta *y(k-n)* donde *n* es el orden de la función de transferencia, son denominados coeficientes *k* seguidos de un número natural desde 1 hasta *2n+1*. Para las funciones de transferencia discretas, estos coeficientes *k* ya son los calculados aquí. Sin embargo en las funciones de transferencia continuas se deben obtener por analogía de las transformaciones realizadas anteriormente con lo obtenido en este apartado.

10.3. Cálculos de la placa de acondicionamiento

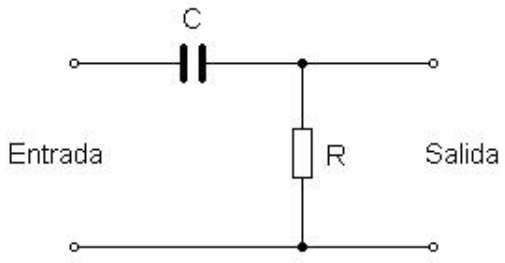


Figura 10.3.0.2 – Filtro RC

En Arduino lo más próximo a un valor analógico son las señales PWM que se pueden obtener por algunos de sus pines. Sin embargo al usar una tarjeta de adquisición de datos (ya sea por medio de un Arduino, otras tarjetas ó DAQ's específicas) no se puede muestrear directamente esta señal ya que el valor leído serían 5v ó 0v dependiendo del estado de la señal PWM en ese preciso instante de muestreo. Esto hace necesaria la inclusión de un filtro paso bajo para la señal de salida del equipo. De esta forma se asegura que la señal analógica de salida puede ser correctamente muestreada por un sistema DAQ.

Sin embargo, se han implementado dos filtros en lugar de uno. Esta decisión ha sido tomada considerando que el equipo está enfocado a ser utilizado por un Arduino como tarjeta de adquisición de datos. Como se dijo anteriormente, un Arduino no puede consegir señales puramente analógicas sino que sólo puede obtener señales PWM. De esta forma este segundo filtro servirá para filtrar una señal PWM que provenga directamente de un Arduino u otro dispositivo similar. Además con la inclusión del filtro para la entrada se podrá usar otro Arduino para introducir la entrada sin tener que añadir circuitería externa para filtrar la señal PWM que genera.

El tipo de filtro a implementar será un filtro paso bajo. Se emplea el filtro paso bajo pasivo RC ya que la señal PWM es muy fácil de filtrar puesto que sólo posee frecuencias muy bajas ó nulas y frecuencias muy altas que son facilmente eliminables. Sin embargo no se debe escoger una frecuencia de corte muy baja ya que añadiría retardos al sistema. Se debe llegar a una solución de compromiso en la que el retardo no sea muy acusado y se atenúen lo máximo posible las frecuencias altas. Finalmente se escogió una frecuencia de corte del filtro de 1 kHz que está a más de una década de la frecuencia de la señal PWM que se maneja en el Arduino (32 kHz aprox.). Para poder utilizar correctamente el filtro para la señal de entrada proveniente directamente de un Arduino se deberá cambiar la frecuencia de la señal PWM que traen por defecto. La frecuncia de la señal PWM de un Arduino, por defecto, es 490 Hz. Para este filtro no sería una frecuencia suficiente por lo que lo recomendable sería establecerla en 32 kHz aproximandamente (la máxima del Arduino).

Los cálculos necesarios para el diseño de dichos filtros se componen simplemente de la fórmula de la frecuencia de corte (f_c) de dicho filtro en función de los componentes utilizados, la resistencia (R) y la capacidad del condensador (C).

$$f_c = \frac{1}{2\pi RC} \tag{10.3.0.5}$$

Se fija el valor del condensador en 100 nF con lo cual se obtiene que el valor de la resistencia deberá ser de:

$$R = \frac{1}{2\pi C} = \frac{1}{2\pi \cdot 100 \cdot 10^{-9}} = 1591{,}55\Omega \tag{10.3.0.6}$$

De esta forma se escoge el valor de resistencia comercial más próximo que es 1500Ω. Si se vuelve a calcular la frecuencia de corte real del filtro se obtiene:

$$f_c = \frac{1}{2\pi 1500 \cdot 100 \cdot 10^{-9}} = 1061 Hz. \tag{10.3.0.7}$$

De esta forma se observa que la frecuencia de corte resultante es más que suficiente para atenuar la frecuencia a la que funcionará la señal PWM generada por la tarjeta

Arduino ($32kHz = 2{,}01 \cdot 10^5 rad/s$) ya que la atenuación se va hasta cerca de 30dBs como se puede ver en la gráfica de la figura 10.3.0.3:

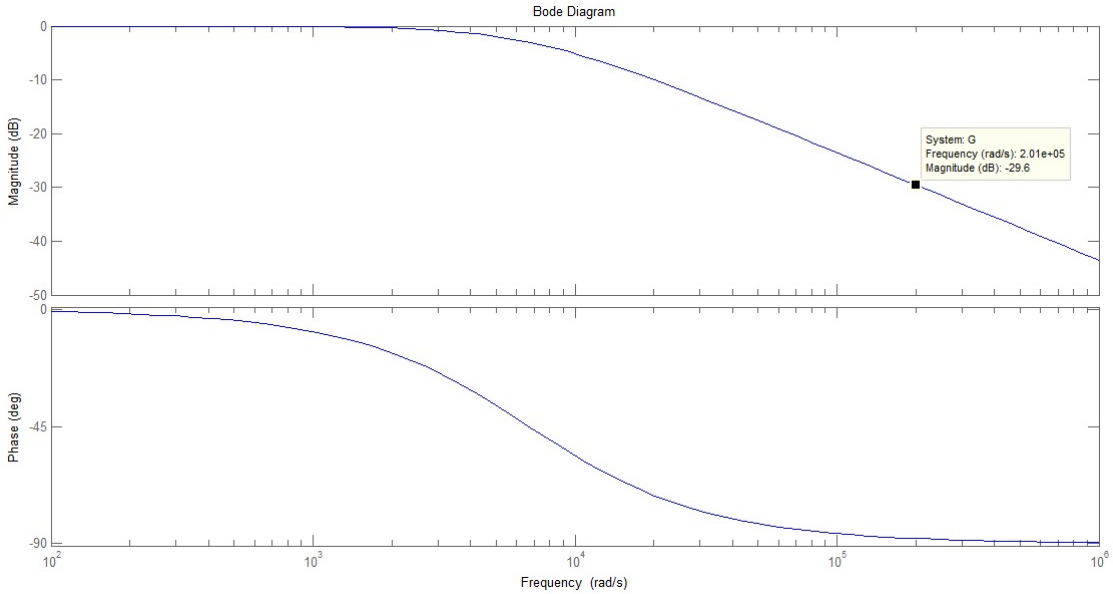


Figura 10.3.0.3 – Diagrama de Bode

10.4. Cálculos de la fuente de alimentación

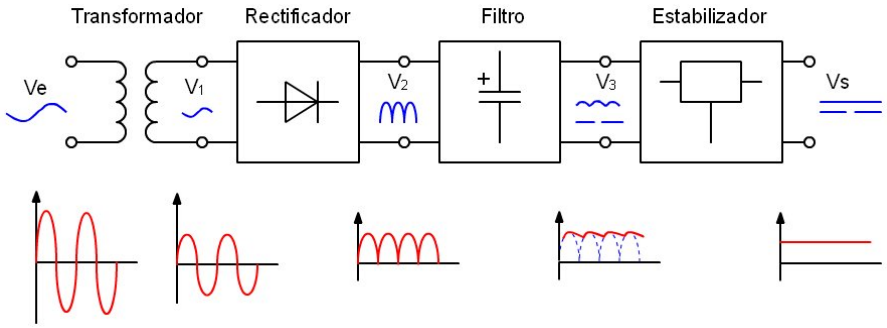


Figura 10.4.0.4 – Fuente de alimentación lineal

La fuente de alimentación necesaria para el Arduino debe proporcionar una tensión entre 7v y 12v. Se ha elejido realizarla de 9v para no sobrecalentar demasiado el regulador de 5v del Arduino. La corriente necesaria para el funcionamiento del equipo estaría formada por unos 50mA que como máximo consumirá la placa Arduino Mega 2560 más los 280mA que consume el display LCD. Observando los reguladores comerciales se ve que no es más económico realizar una fuente de alimentación de menos de 1A. Por lo tanto se va a dimensionar una fuente de alimentación de 9v y de 1A.

10.4.1. Regulador

Se empleará el regulador LM7809 que proporciona una tensión lineal de 9v y una corriente máxima de 1A. Para los siguientes cálculos se sobredimensionará esta corriente un 50 % a modo de margen de seguridad. Es decir la corriente máxima será de 1,5A.

10.4.2. Puente rectificador

Se comienza por la selección del puente rectificador ya que para la selección del transformador se debe conocer la caída de tensión que se producirá en los diodos del puente rectificador.

El puente rectificador debe soportar la corriente máxima que demandará la fuente. Es decir, 1,5A. Observando catálogos se ve que el puente rectificador B40-C1500 soporta una corriente en régimen permanente de hasta 1,5A lo cual cumple las especificaciones necesarias. La máxima tensión que soporta este componente son 40v, dato suficiente si se van a obtener tensiones de salida de 9v. Además la caída de tensión en este componente será de 2v como indica la hoja de características del mismo. Con estos datos se calcula el transformador necesario para nuestra fuente de alimentación.

10.4.3. Transformador

El transformador debe cumplir las siguientes especificaciones:

- Tensión de salida suficiente.
- Corriente máxima admisible sin sobrecalentamiento.

Para el cálculo de la tensión del secundario del transformador (V_{ac}) se debe considerar los siguientes elementos:

La tensión continua final que se desee obtener (V_{out}), la caída de tensión en el regulador (V_{reg}), la caída de tensión en el rectificador (V_{rect}), tensión del rizado del filtro por condensador (V_{riz}) y la eficiencia del transformador (η).

Se va a calcular la tensión del secundario del transformador partiendo de que la tensión de salida que se desee obtener es de 9v y que el “dropout”del mismo es de 2v. Inicialmente se supone una tensión de rizado a la entrada del regulador de 3v. Es decir, se parte de los siguientes datos:

- V_{out} =9v
- V_{reg} =2v
- V_{rect} =2v
- V_{riz} =3v
- η =0,95

$$V_{ac} = \frac{V_{out} + V_{reg} + V_{rect} + V_{riz}}{\eta} \times \frac{1}{\sqrt{2}} \tag{10.4.3.1}$$

Sustituyendo los valores anteriores en 10.4.3.1, se obtiene:

$$V_{ac} = \frac{9 + 2 + 2 + 3}{0,95} \times \frac{1}{\sqrt{2}} = 11,91v \tag{10.4.3.2}$$

Se debe escoger un transformador del valor comercial inmediatamente superior, es decir, V_{ac}= 12v. Por otra parte, debe asegurarse de que el transformador suministre la corriente máxima sin un sobrecalentamiento excesivo. Por lo cual se debe seleccionar el transformador de valor comercial inmediatamente superior a la corriente máxima demandada, es decir 1,5A.

10.4.4. Condensadores de filtro

En el análisis aproximado de un filtro por condensador, en un circuito de doble onda, se obtiene la fórmula 10.4.4.1 para el cálculo del condensador, en función de la corriente continua de carga y la tensión de rizado:

$$C = \frac{I_{dc}}{2 \cdot f \cdot V_r} \tag{10.4.4.1}$$

Sin embargo, experimentalmente se comprobó que no era del todo exacta, obteniéndose la que a continuación se indica y que será la que se utilizará en los diseños:

$$C = \frac{I_{dc}}{V_r} \cdot 7,2 \cdot 10^{-3} \tag{10.4.4.2}$$

Sustituyendo los datos en la fórmula 10.4.4.2:

$$C = \frac{1}{3} \cdot 7,2 \cdot 10^{-3} = 2400\mu F \tag{10.4.4.3}$$

Se debe escoger el condensador de valor comercial inmediatamente superior. Se empleará entonces un condensador de 3300μF. Con este condensador se asegura una

tensión mínima a la entrada del regulador, superior a la necesaria.

Se emplearán también los condensadores recomendados por el fabricante del regulador. Éstos están situados a la entrada y a la salida del del mismo a modo de filtro próximo de entrada y filtro de salida para prevenir variaciones en el consumo.

10.4.5. Cálculo de la resistencia del LED

La fuente de alimentación llevará un LED (Light Emitting Diode, Diodo emisor de luz) de encendido de color verde. Para calcular la resistencia en serie necesaria se debe emplear la siguiente fórmula:

$$R = \frac{V - V_{LED}}{I} \tag{10.4.5.1}$$

Donde:

- V es la tensión de alimentación a la que va ir conectado
- V_{LED} es la caída de tensión en el LED (Hoja de características)
- I es la corriente que circulará por el LED.

Al tratarse de un LED indicador, una corriente de 10mA es suficiente. La caída de tensión en el LED es de 2,1v. La tensión de alimentación serán 9v (procedientes del regulador):

$$R = \frac{9 - 2{,}1}{10 \cdot 10^{-3}} = 690\Omega \tag{10.4.5.2}$$

Se utiliza la resistencia de valor comercial más cercano, es decir, R=680Ω.

10.4.6. Cálculo del fusible

Para el cálculo del fusible se utiliza un factor multiplicador por 2 para prevenir los picos de corriente que se forman el en primario del transformador en el momento del encendido. Entonces para el primario, se obtiene:

$$I = 1 \cdot \frac{12}{230} \cdot 2 \cdot 1{,}5 = 156mA \tag{10.4.6.1}$$

Se empleará un fusible de 500mA para el primario ya que es el valor comercial más cercano y más habitual que se puede encontrar.

Capítulo 11

Anexo 3: Comprobación de resultados

Este anexo está dedicado a la comprobación de resultados obtenidos con el equipo. Se realizan dos pruebas. Por un lado se comprueba la constancia del periodo de muestreo y, por otra parte, se contrastan los resultados de las simulaciones con los obtenidos con Matlab.

11.1. Comprobación del periodo de muestreo

Para comprobar el periodo de muestreo se emplea un osciloscopio y un punto de prueba en el Arduino. Para ello se necesitan las siguientes líneas de código en la función setup del Arduino. En ellas se declara el pin digital 2 del Arduino como salida y se inicia su valor a cero.

```
1  // Punto de prueba para comprobar el periodo de muestreo
2  pinMode (2,OUTPUT);
3  digitalWrite(2,LOW);
```

A continuación se hará que el pin digital 2 cambie de valor cada vez que se escriba un nuevo valor de salida PWM en el pin analógico 9 del Arduino. Para conseguirlo se hizo que al final de la subrutina de interrupción del timer (y no al principio para tener en cuenta el tiempo de ejecución de las instrucciones intermedias) se invierte el valor del pin digital 2. Para ello se escribe la siguiente línea de código:

```
1  digitalWrite(2,!digitalRead(2));
```

Para situarse en el peor caso posible se eligió simular una función de transferencia continua (periodo de muestreo mínimo: 10ms), de orden 3 y con el envío de datos por el puerto serie activado. Estas serían las condiciones más desfavorables para que el periodo de muestreo no fuera constante o no fuera el que se ha indicado previamente. Sin embargo observando el código se comprueba que la secuencia de operaciones a

ejecutar es siempre la misma, con lo cual le llevará siempre el mismo tiempo hacerla. Al final los resultados han sido satisfactorios pues el periodo de muestreo se cumple siempre y se mantiene constante. Tan sólo existe una pequeña variación aleatoria de unos microsegundos que se considera irrelevante cuando se trata con un periodo de muestreo 1000 veces superior (10 ms).

A continuación se muestran unas imágenes de resultados obtenidos probando diferentes periodos de muestreo y en todas ellas se ve que se cumple a la perfección. En el osciloscopio se activan las opciones de Periodo (que será el doble del que se introduce al tratarse de una señal cuadrada), ancho de pulso (que deberá coincidir con el periodo de muestreo introducido) y “duty cicle”para comprobar que la señal es totalmente cuadrada lo que indicará que el periodo de muestreo se mantiene constante entre una muestra y la siguiente:

- Periodo de muestreo de 10ms (figura 11.1.0.1).

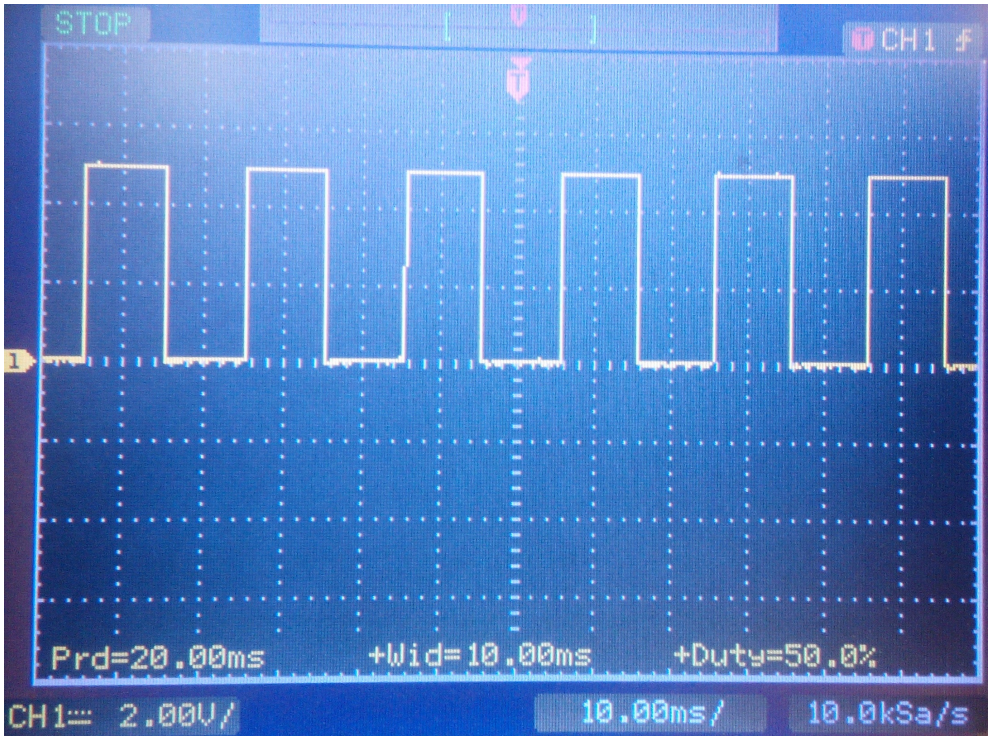


Figura 11.1.0.1 – Periodo de muestreo=10ms

- Periodo de muestreo de 50ms (figura 11.1.0.2).

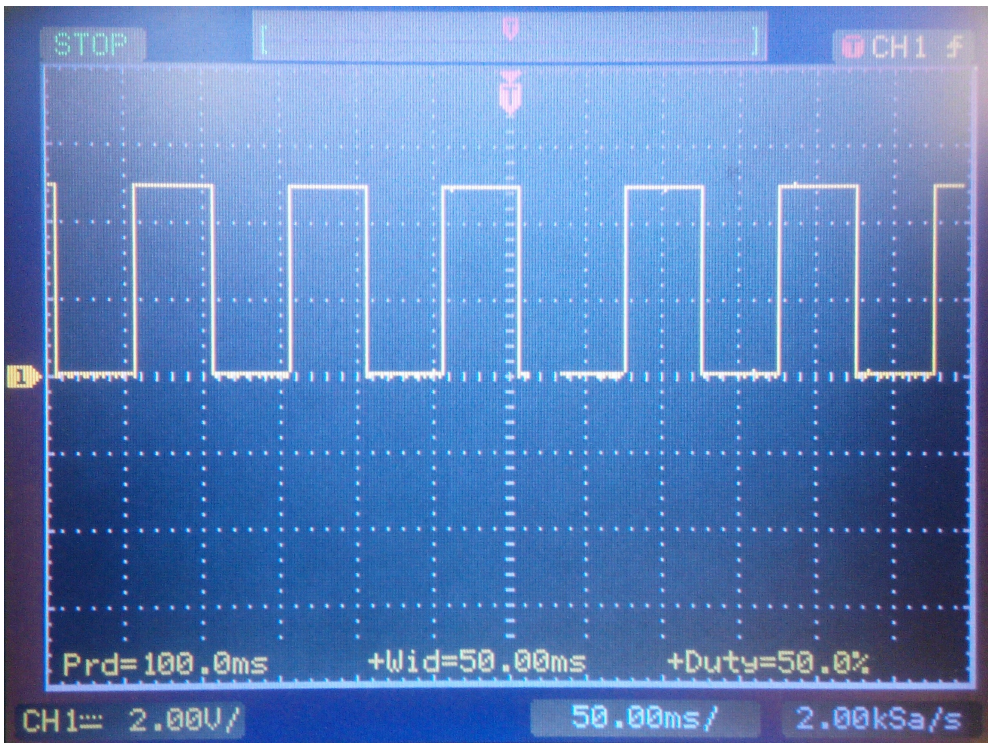


Figura 11.1.0.2 – Periodo de muestreo=50ms

- Periodo de muestreo de 100ms (figura 11.1.0.3).

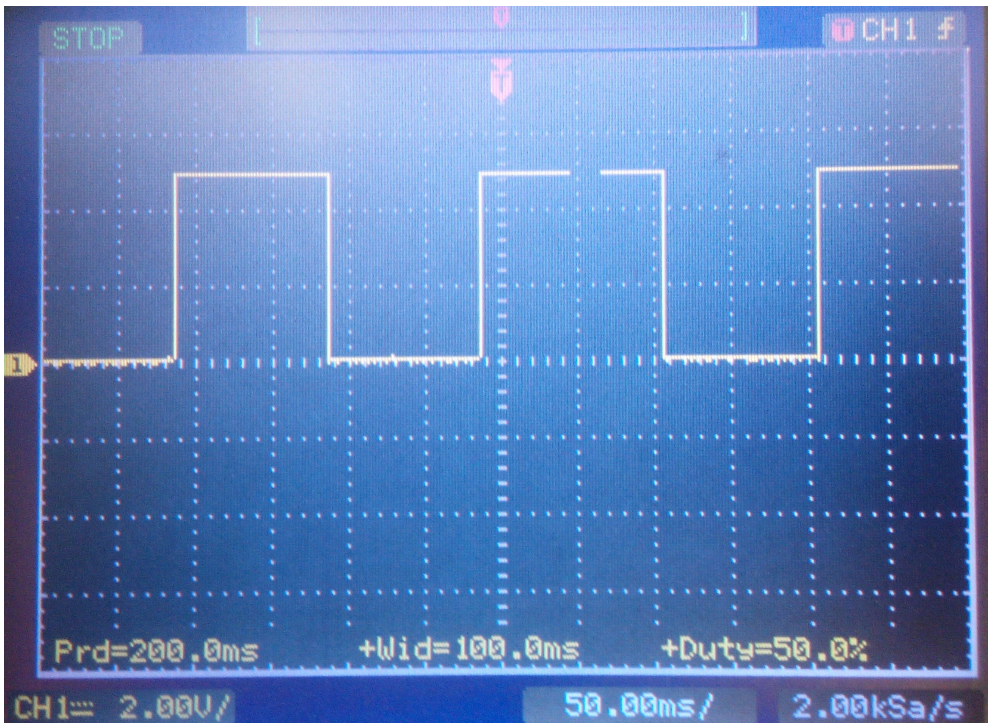


Figura 11.1.0.3 – Periodo de muestreo=100ms

- Periodo de muestreo de 1s (figura 11.1.0.4).

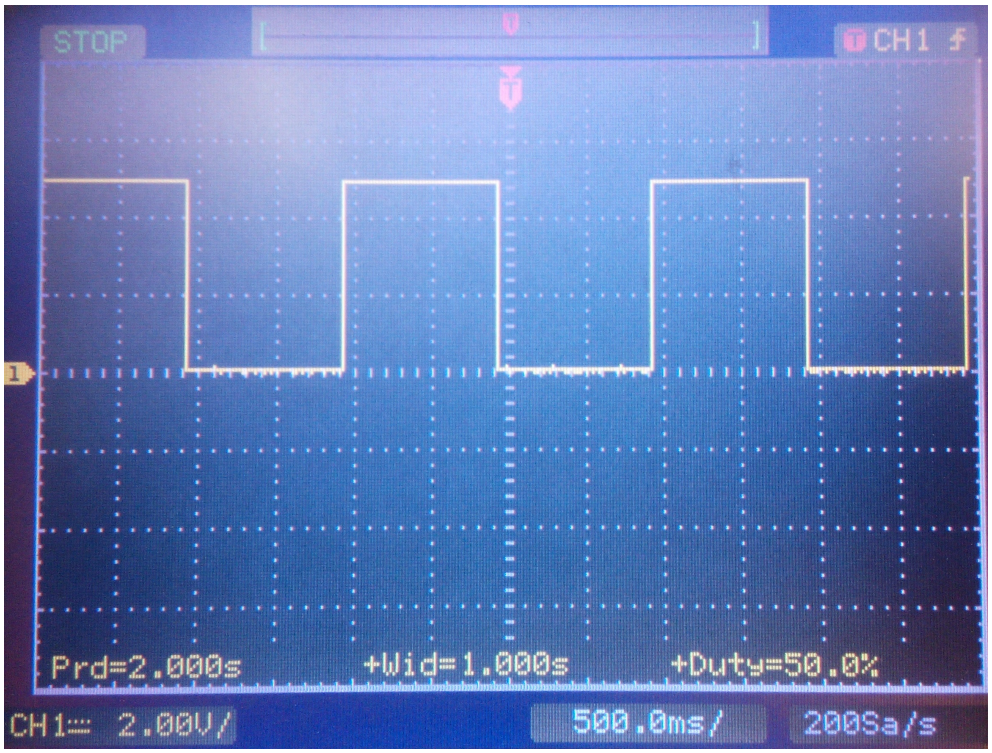


Figura 11.1.0.4 – Periodo de muestreo=1s

11.2. Comprobación de resultados

En este apartado se procede a comprobar los resultados obtenidos con el emulador comparándolos con los que se obtendrían simulando la misma función de transferencia en Matlab. Para ello se empleará una señal de entrada de simulación que en este caso será un escalón unitario. Para que las comprobaciones puedan ser válidas, éstas se realizan con varias funciones de transferencia.

Para comprobar el funcionamiento se utilizarán funciones de transferencia continuas puesto que si éstas funcionan correctamente, las funciones de transferencia discretas también lo harán. Esto se debe a que las primeras, antes de poder simularlas se discretizan y se convierten a discretas.

11.2.1. Función de transferencia de orden 1

En primer lugar se muestra el *script* utilizado en matlab para realizar la simulación de las funciones de transferencia. Se trata de un *script* básico de simulación:

```
N = [1];
D = [1  1];
[Nz,Dz] = c2dm (N,D,0.01,'tustin')
y = dstep (Nz,Dz);
plot (y,'.');
title ('Respuesta escalón de un sistema discreto');
xlabel ('Periodo de muestreo');
ylabel ('Salida');
grid;
```

En el *script* anterior tan sólo se introduce el numerador y el denominador de la función de transferencia continua que se desee simular. Se crea la función de transferencia con la función *tf*. Una vez realizado esto se convierte a numerador y denominador de una función de transferencia discreta con la función *c2dm* a la que se le debe indicar la función de transferencia continua anterior, el periodo de muestreo (10ms) y el tipo de aproximación empleada. Una vez hecho esto tan sólo se aplica un escalón unitario discreto y se plotean los resultados. Para esta determinada función de transferencia básica los resultados obtenidos han sido los mostrados en la figura 11.2.1.1.

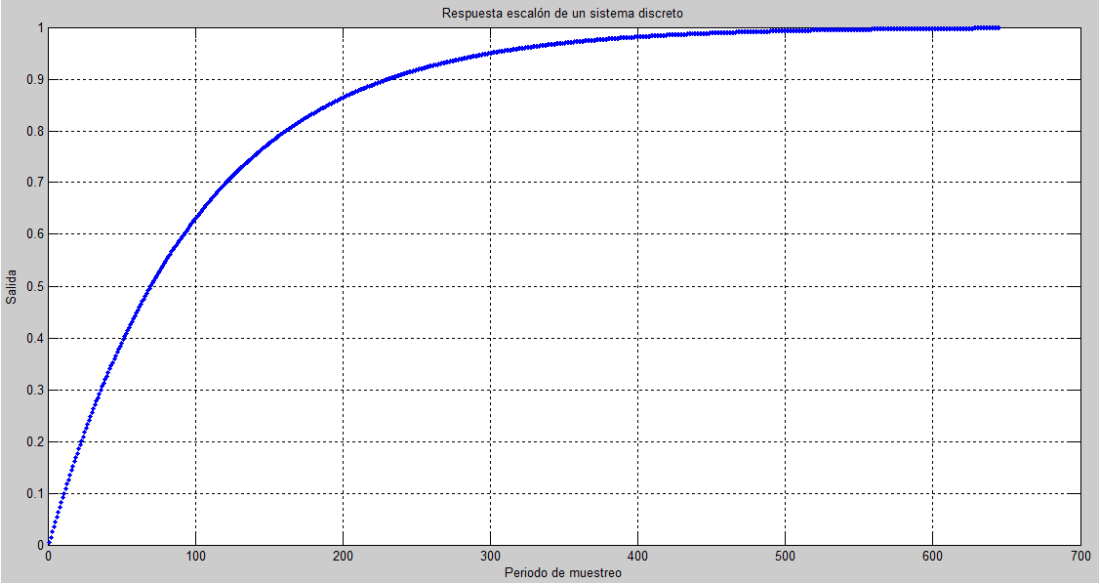


Figura 11.2.1.1 – Gráfica de simulacion Matlab 1

Utilizando el equipo emulador para simular la misma función de transferencia con el mismo periodo de muestreo y el mismo método de aproximación resulta la gráfica mostrada en la figura 11.2.1.2.

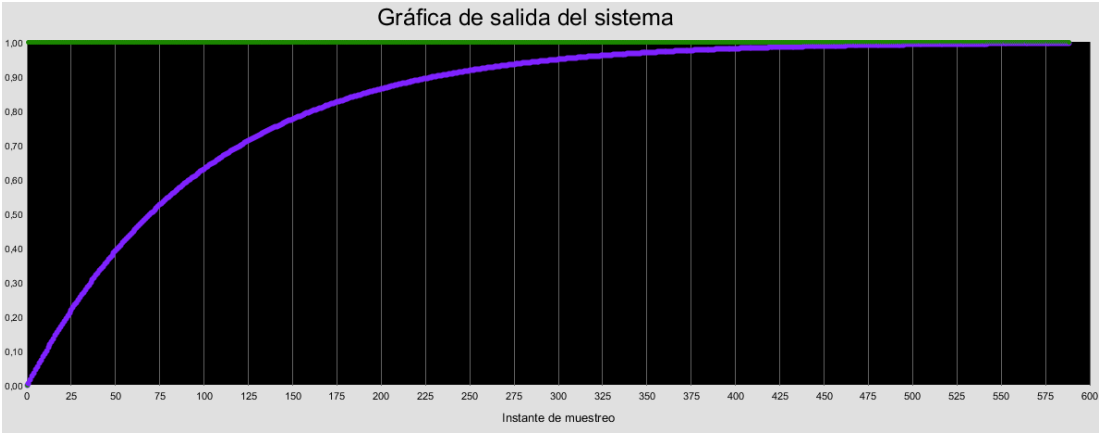


Figura 11.2.1.2 – Gráfica de simulacion 1

Esta simulación es la obtenida del equipo emulador por medio de la comunicación serie del equipo y por medio de un *sketch* en el programa Processing para visualizar los datos. Se puede concluir que los datos son idénticos en cada instante de muestreo a los obtenidos en el software de simulación de Matlab.

Para ver mejor esta identidad se consultan los primeros valores de la salida Y obtenida en matlab y los primeros valores obtenidos por el puerto serie del Arduino. Se puede ver que los resultados son idénticos en figura 11.2.1.3:

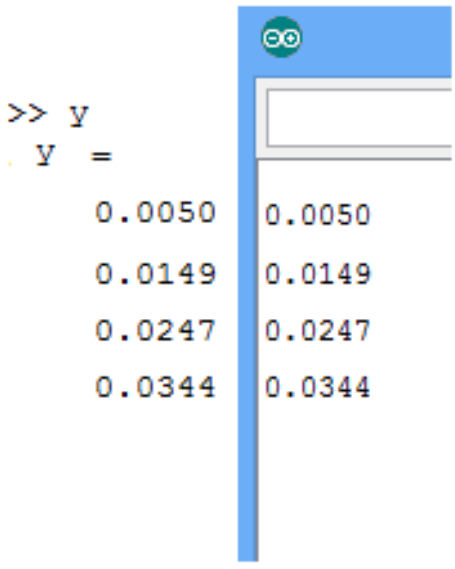


Figura 11.2.1.3 – Datos simulacion 1

11.2.2. Función de transferencia de orden 3

Para comprobar el funcionamiento del equipo para funciones de transferencia más complejas se ha llevado a cabo la simulación de una función de transferencia de orden tres. El script de Matlab es el mismo que el anterior pero ahora cambian los coeficientes del numerador y del denominador. Los coeficientes que se emplean en este caso son los de la función de transferencia tipo amortiguador-carro-resorte incluida en el equipo (ver código siguiente). En la figura 11.2.2.1 se muestra el resultado obtenido.

```
N = [1];
D = [1 4 5 2];
[Nz,Dz] = c2dm (N,D,0.01,'tustin')
y = dstep (Nz,Dz);
plot (y,'.');
title ('Respuesta escalón de un sistema discreto');
xlabel ('Periodo de muestreo');
ylabel ('Salida');
grid;
```

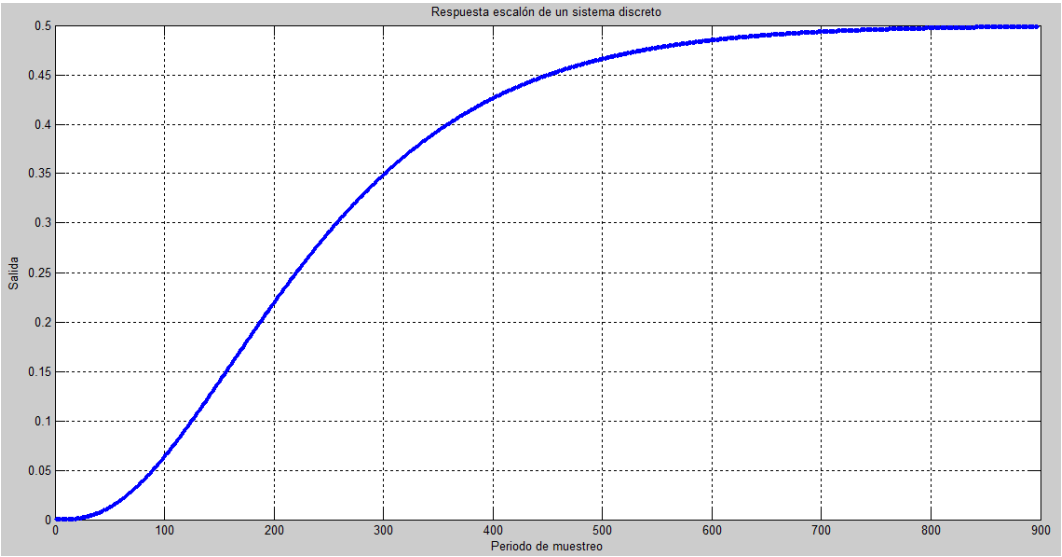


Figura 11.2.2.1 – Gráfica de simulacion Matlab 2

Utilizando el equipo emulador para simular la misma función de transferencia con el mismo periodo de muestreo y el mismo método de aproximación resulta la gráfica de la figura 11.2.2.2.

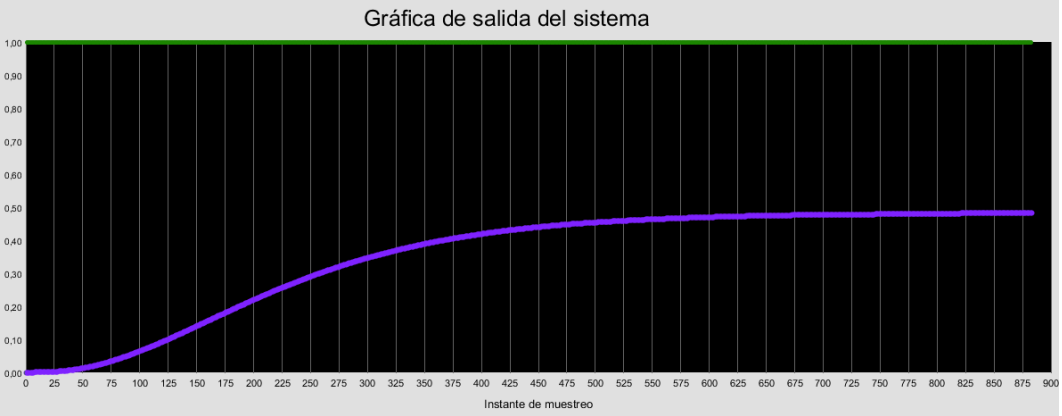


Figura 11.2.2.2 – Gráfica de simulacion 2

Se puede concluir que los datos son idénticos en cada instante de muestreo a los obtenidos en el software de simulación de Matlab. Para ver mejor esta identidad se consultan los primeros valores de la salida Y obtenida en matlab y los primeros valores obtenidos por el puerto serie del Arduino. Se puede ver que los resultados son idénticos en la figura 11.2.2.3.

```
>> y

y =

    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0001
    0.0001
    0.0001
    0.0002
    0.0002
    0.0003
    0.0004
    0.0004
    0.0005
    0.0006
    0.0008
    0.0009
    0.0010
    0.0012
```

```
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0001
0.0001
0.0001
0.0002
0.0002
0.0003
0.0004
0.0004
0.0005
0.0006
0.0008
0.0009
0.0010
0.0012
```

Figura 11.2.2.3 – Datos simulacion 2

Con estas dos comprobaciones de resusltados, se puede ver el alto grado de precisión del equipo. Por un lado se comprueba que el periodo de muestreo es el correcto y por otra parte que los resultados de las funciones de transferencia son exactamente iguales a los obtenidos en el software de Matlab.

Esta pruebas ha servido para demostrar que el equipo calcula de forma correcta los valores que debe ir tomando la señal de salida ante una determinada función de entrada. Además se comprueba que lo hace bien en el transcurso del tiempo, es decir, que el periodo de muestreo es el indicado y que es siempre constante.

Capítulo 12

Anexo 4: Manual de usuario y especificaciones

El en presente anexo se desarrolla el manual para la correcta utilización del dispositivo emulador. En primer lugar se indican las instrucciones de uso paso a paso para poder interactuar con el equipo y por otra parte se explican todas las conexiones necesarias para que el funcionamiento del equipo sea el correcto.

Este anexo servirá de guía de uso del equipo donde se mostrarán los resultados obtenidos al utilizarlo y finalmente se incluirá una tabla con las prestaciones y especificaciones del equipo emulador.

12.1. Puesta en marcha del equipo

En primer lugar se debe alimentar el equipo para que pueda funcionar. La alimentación podría obtenerse directamente de los puertos USB de un ordenador. Esto implicaría disponer siempre de un ordenador ó cualquier otro equipo con salida USB que dé la corriente suficiente. Sin embargo, con el diseño de una fuente de alimentación externa sólo se debe enchufarla a la toma de corriente eléctrica de la red y, finalmente, se debe enchufar el conector macho de la fuente de alimentación en la placa Arduino Mega. De esta forma el equipo quedará totalmente alimentado.

Cabe mencionar que la alimentación por el puerto USB también sería posible siempre y cuando la fuente conectada proporcione una corriente de al menos 300mA. En caso de conectar ambas alimentaciones (USB y conector de alimentación) el Arduino seleccionará automáticamente la segunda. De esta forma el puerto USB se podrá seguir usando igualmente para la transmisión de datos por el puerto serie.

12.2. La pantalla de inicio

Una vez conectado el dispositivo, el display LCD debería iluminarse y presentar una pantalla de inicio similar a la que se puede ver en la imagen 12.2.0.1:

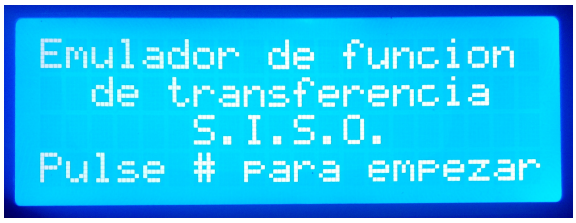


Figura 12.2.0.1 – Pantalla de inicio

Una vez en este estado se tienen dos opciones principales. Estas dos opciones estan disponibles pulsando la tecla asterisco [*] ó la tecla almohadilla [#] del teclado matricial. En el primer caso se entraría en las opciones de configuración del dispositivo que, como se verá más adelante, se trata sobre la configuración del puerto serie y del ruido de la señal de salida. En el segundo caso se accede directamente a la elección de la función de transferencia a emular. En este caso las opciones de configuración del puerto serie y del ruido de la señal de salida serán las que están por defecto: El puerto serie desactivado y el ruido de la señal de salida nulo.

12.3. La pantalla de configuración

Al pulsar la tecla [*] se presenta la siguiente pantalla:

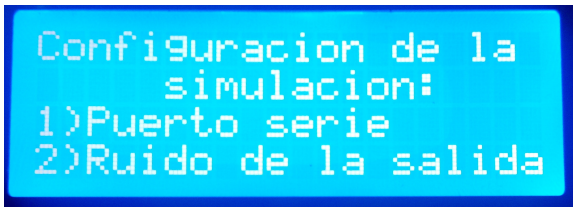


Figura 12.3.0.2 – Pantalla de configuracion

En este punto se tienen tres teclas con las que se puede interacturar con el dispositivo. Para cada una de ellas se irá a un menú de configuración diferente según la tecla pulsada, tal y como se indica a continuación:

- Tecla [1]: Para configurar el puerto serie.

- Tecla [2]: Para configurar el ruido de la señal de salida.
- Tecla [A]: Regresa al menú anterior, en este caso al menú principal.

12.3.1. Configuración del puerto serie

En caso de elegir configurar el puerto serie, se tendrán dos opciones de configuración. En el LCD aparecerá una pantalla similar a la siguiente:

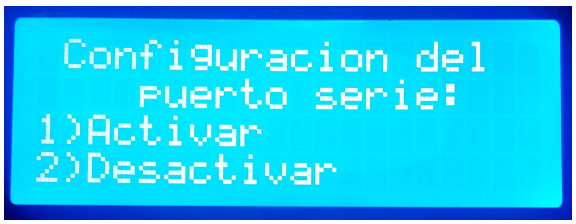


Figura 12.3.1.1 – Pantalla de configuracion serie

- Tecla [1]: Para activar el envío de información por el puerto serie.
- Tecla [2]: Para desactivar el envío de información por el puerto serie.
- Tecla [A]: Regresa al menú anterior, en este caso al menú de configuración.

En caso de activar el puerto serie para poder visualizar ó representar los datos de la simulación se deben tener en cuenta algunas precauciones. En el ordenador al que se conecte el dispositivo debe asegurarse de que seleccione el puerto COM asignado por el ordenador al Arduino Mega 2560. Por ejemplo, en Matlab simplemente se debe indicar el número del puerto COM que el ordenador le asigna y la velocidad de sincronismo que en este caso debe ser 115200 baudios. Sin embargo en el *sketch* de simulación realizado en Processing lo que se debe seleccionar es el índice del vector compuesto por los puertos COM actuales del ordenador. Si no se tiene ningún otro dispositivo conectado que utilice un puerto COM se deberá indicar el índice 0 del vector de puertos en el código fuente del *sketch* de simulación de Processing. Además se debe asegurar que la velocidad de sincronismo sea de 115200 baudios. El vector de puertos COM del ordenador se denomina en *Processing* SerialList.

En caso de desactivar el puerto serie del equipo, para visualizar los datos se deberá hacer de otra forma, por ejemplo, con la tarjeta de adquisición de datos con la que se esté utilizando el equipo.

12.3.2. Configuración del ruido de la señal de salida

En este menú de configuración se puede elegir el ruido de la señal de salida de la función de transferencia. Los porcentajes de ruido parecen muy bajos pero en la práctica se pueden llegar a hacer grandes. Esto se debe a que el ruido aquí introducido es totalmente aleatorio. Además el ruido introducido en la muestra actual ya depende de una muestra previa con su ruido. Esta cadena hace que el ruido real sea muy superior a los porcentajes aquí indicados.

En esta pantalla se tienen cuatro opciones en cuanto al ruido (figura 12.3.2.1:

- Tecla [1]: Sin ruido.
- Tecla [2]: Ruido del 0.005 %.
- Tecla [3]: Ruido del 0.01 %.
- Tecla [4]: Ruido del 0.1
- Tecla [A]: Regresa al menú anterior, en este caso al menú de configuración.

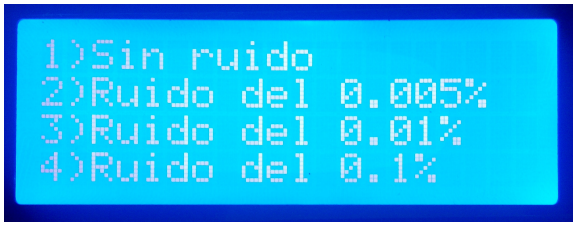


Figura 12.3.2.1 – Pantalla de configuracion del ruido

Estando en esta pantalla se debe pulsar dos veces la tecla [A] para volver al menú principal del dispositivo. En el menú principal, como se mencionó anteriormente, se pulsará la tecla [#] para comenzar con la configuración ó elección de una determinada función de transferencia.

12.4. Pantalla de selección de la función de transferencia

En esta pantalla se presentan las distintas opciones que tiene el dispositivo en cuanto al tipo de función de transferencia que se quiera emular. En un principio aparecerá una pantalla como la mostrada en la figura 12.4.0.2:

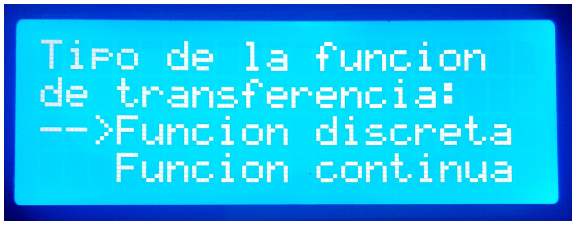


Figura 12.4.0.2 – Pantalla de selección 1

La flecha funciona como un cursor de selección, es decir, marca la opción seleccionada en cada instante. Este menú es un menú desplazable en el que se puede desplazar el cursor para seleccionar el tipo de función de transferencia que se desee emular.

En esta pantalla hay cuatro teclas con las que se puede interactuar. Pulsar cualquiera otra no tendrá ningún efecto. Son las siguientes:

- Tecla [#]: Elige la opción marcada por la flecha.
- Tecla [C]: Desplazar el cursor del menú hacia arriba.
- Tecla [D]: Desplazar el cursor del menú hacia abajo.
- Tecla [A]: Regresa al menú anterior, en este caso al menú principal

De este modo, si pulsa una vez la tecla [D] la pantalla cambiaría a la mostrada en la figura 12.5.1.1:

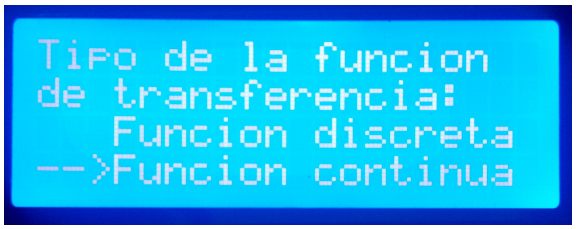


Figura 12.4.0.3 – Pantalla de selección 2

Como se dijo anteriormente para seleccionar una determinada opción se debe situar la flecha en ella y pulsar la tecla [#]. Todas estas posibilidades se pueden resumir en tres claramente diferenciadas:

1. Función de transferencia discreta. Se corresponde con la primera opción. El equipo pedirá posteriormente el orden de la función de transferencia a emular, los parámetros del numerador y del denominador de dicha función de transferencia en Z y periodo de muestreo de la misma.

2. Función de transferencia continua. Se corresponde con la segunda opción. El equipo pedirá posteriormente el tipo de aproximación a ejecutar, el orden de la función de transferencia a emular y los parámetros del numerador y del denominador de dicha función de transferencia en *s*.

3. Resto de funciones de transferencia. El resto de las funciones de transferencia se corresponde ya con una función continua determinada, es decir, el equipo tiene preconfiguradas unas funciones estándar de sistemas reales. El tipo de aproximación utilizada para éstas es la aproximación de Tustin ya que es más precisa y real. Una vez elegida una de estas funciones de transferencia el equipo comenzará a emularlas de forma instantánea.

12.5. Configuración de una función de transferencia discreta

Si se elije esta opción, el equipo pedirá de forma inmediata el orden de dicha función de transferencia. Aparecerá la pantalla mostrada en la figura 12.5.1.1:

12.5.1. Orden de la función de transferencia discreta

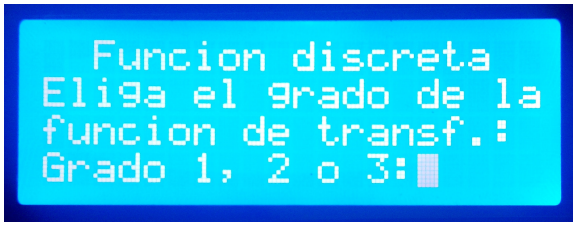


Figura 12.5.1.1 – Pantalla de selección del orden

En esta pantalla sólo se puede interactuar con el equipo con cuatro teclas:

- Tecla [1]: Función de transferencia de orden 1.
- Tecla [2]: Función de transferencia de orden 2.
- Tecla [3]: Función de transferencia de orden 3.
- Tecla [A]: Regresa al menú anterior, en este caso al menú de elección de la función de transferencia.

12.5.2. Parámetros de la función de transferencia discreta

Una vez introducido el orden de la función, el equipo comenzará a pedir los parámetros de la función de transferencia. Se comienza por los coeficientes del numerador. El equipo pedirá los coeficientes desde el grado introducido anteriormente hasta el grado cero. La pantalla de introducción de los coeficientes del numerador es la mostrada en la figura 12.5.2.1 (se ha elegido previmente orden 2):

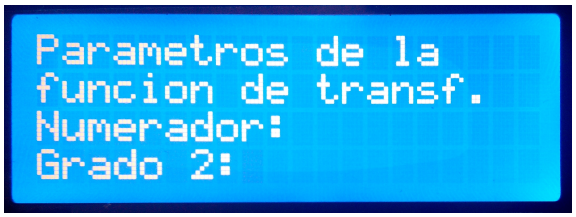


Figura 12.5.2.1 – Pantalla coeficientes numerador

En este punto se debe interactuar con el equipo con las teclas numéricas, la tecla [*], la tecla [C] que servirá para introducir un signo negativo antes de pulsar cualquier tecla y la tecla [#] que será aceptar el parámetro introducido.

- Tecla [C]: Introduce un signo negativo.
- Tecla [*]: Punto decimal
- Tecla [#]: Aceptar el parámetro introducido.
- Tecla numéricas: Para introducir el valor de cada coeficiente.

Una vez introducidos los parámetros del numerador, el equipo pedirá que se introduzcan los del denominador. La introducción de los datos se realizará de la misma forma que para los coeficientes del numerador. La pantalla que aparecerá se muestra en la figura 12.5.2.2

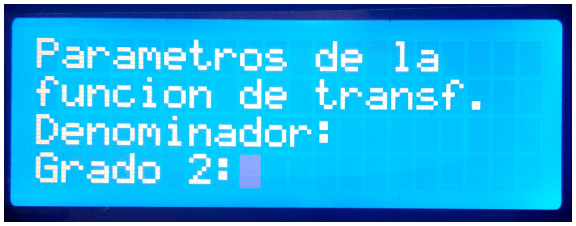


Figura 12.5.2.2 – Pantalla coeficientes denominador

12.5.3. Periodo de muestreo

Una vez introducida la función de transferencia el equipo solicitará la introducción del periodo de muestreo. La introducción de los datos se realiza de la misma forma que antes. La pantalla de introducción del periodo de muestreo es la la que aparece en la figura 12.5.3.1:

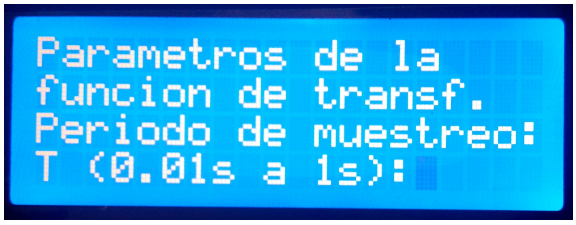


Figura 12.5.3.1 – Pantalla periodo de muestreo

El periodo de muestreo debe ser introducido en segundos y su valor deberá estar entre 0.01s y 1s. Para valores inferiores a 0.01s el equipo tomará 0.01 segundos como periodo de muestreo. Para valores superiores a 1s el equipo tomará 1s como periodo de muestreo.

12.5.4. Simulación de la función de transferencia discreta

Al terminar de introducir el periodo de muestreo y pulsar la tecla [#], el equipo comenzará a simular la función de transferencia de forma instantánea. En este caso se mostrarán dos posibles pantallas en caso de tener activada o no la opción de envío de los datos de la simulación por el puerto serie. Son las figuras 12.5.4.1 y 12.5.4.2.

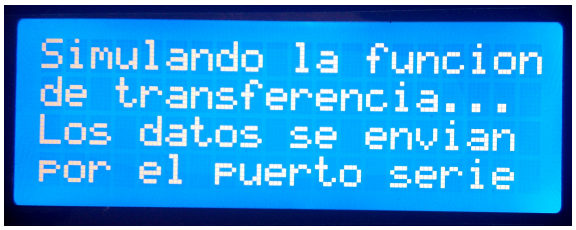


Figura 12.5.4.1 – Pantalla simulacion con puerto serie

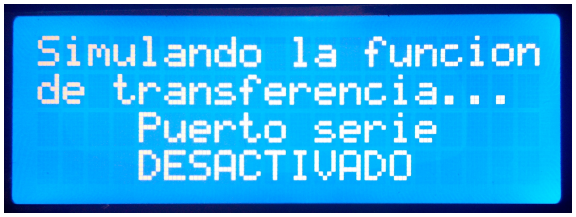


Figura 12.5.4.2 – Pantalla simulacion sin puerto serie

Se debe tener la precaución de tener la entrada conectada y a cero en el momento de comenzar la simulación. De esta forma la señal de salida no variará hasta que la entrada sea distinta de cero.

12.6. Configuración de una función de transferencia continua

Para el caso de elegir la emulación de una función de transferencia continua, el proceso es similar al anterior. Tan sólo cambian algunas opciones ya que no tiene sentido hablar de periodo de muestreo en una función de transferencia continua. Además se debe seleccionar el tipo de aproximación que interese.

12.6.1. Tipo de aproximación

Al elegir emular una función de transferencia continua la primera pantalla que mostrará el LCD será la de la figura 12.6.1.1, donde pide el tipo de aproximación que se quiera utilizar:

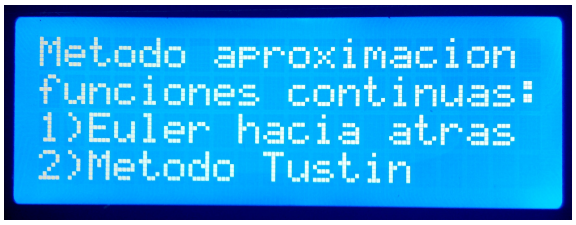


Figura 12.6.1.1 – Pantalla seleccion aproximacion

En esta pantalla hay dos opciones y, como siempre, se debe elegir una de ellas por medio de las teclas [1] y [2]. También existe la posibilidad de volver al menú anterior pulsando la tecla [A].

12.6.2. Orden de la función de transferencia continua

Una vez elegido el tipo de aproximación, el equipo solicitará, al igual que con las funciones de tranferencia discretas, el orden de la función de transferencia continua que se desee emular. La pantalla es la misma que para el caso de la función de transferencia discreta (figura 12.6.2.1:

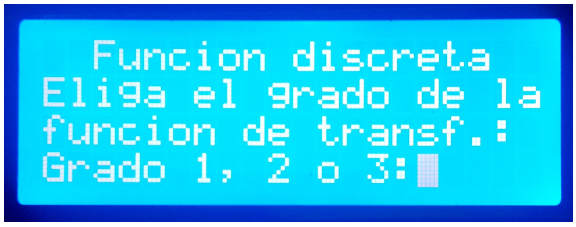


Figura 12.6.2.1 – Pantalla de selección del orden 2

12.6.3. Parámetros de la función de transferencia continua

Una vez introducido el orden de la función, el equipo comenzará a pedir los parámetros de la función de transferencia. Al igual que antes se comienza por los coeficientes del numerador. El equipo pedirá los coeficientes desde el grado introducido anteriormente hasta el grado cero. La pantalla de introducción de los coeficientes del numerador es la de la figura 12.6.3.1:

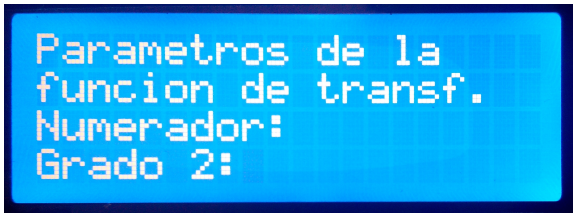


Figura 12.6.3.1 – Pantalla coeficientes numerador 2

En este punto se debe interactuar con el equipo con las teclas numéricas, la tecla [*], la tecla [C] que servirá para introducir un signo negativo antes de pulsar cualquier tecla y la tecla [#] que será aceptar el parámetro introducido.

- Tecla [C]: Introduce un signo negativo.
- Tecla [*]: Punto decimal
- Tecla [#]: Aceptar el parámetro introducido.
- Tecla numéricas: Para introducir el valor de cada coeficiente.

Una vez introducidos los parámetros del numerador, el equipo pedirá que se introduzcan los del denominador. La introducción de los datos se realizará de la misma forma que para los coeficientes del numerador. La pantalla que aparecerá en este caso se puede ver en la figura 12.6.3.2:

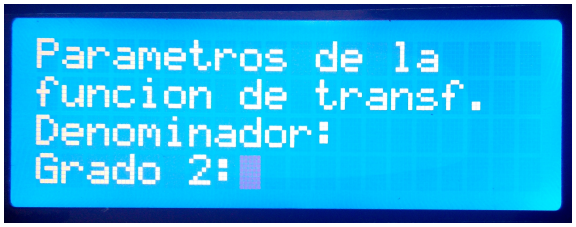


Figura 12.6.3.2 – Pantalla coeficientes denominador 2

12.6.4. Simulación de la función de transferencia continua

Al terminar de introducir el último parámetro del denominador de la función de transferencia el equipo comenzará la emulación. En este caso se mostrarán dos posibles pantallas en caso de tener activada o no la opción de envío de los datos de la simulación por el puerto serie. Son las mostradas en las figuras 12.6.4.1 y 12.6.4.2:

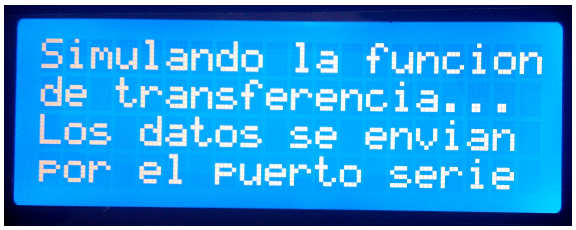


Figura 12.6.4.1 – Pantalla simulacion con puerto serie 2

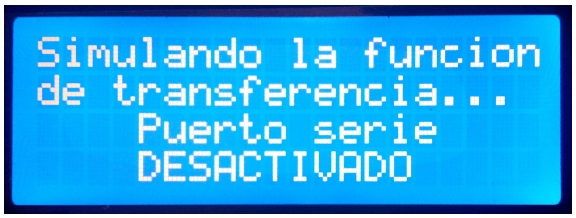


Figura 12.6.4.2 – Pantalla simulacion sin puerto serie 2

Se debe tener la precaución de tener la entrada conectada y a cero en el momento de comenzar la simulación. De esta forma la señal de salida no variará hasta que la entrada sea distinta de cero.

12.7. Simulación de una función de tranferencia tipo

En caso de elegir la emulación de una función de transferencia tipo, el equipo no pedirá la configuración de ningún parámetro. Tan sólo comenzará a emularla con los parámetros preconfigurados de dicha función de transferencia. En este caso también

aparecerán dos posibles pantallas durante la simulación en función de si el puerto serie está activado o no. Al igual que antes se debe tener la precaución de tener la entrada conectada y a cero en el momento de comenzar la simulación. De esta forma la señal de salida no variará hasta que la entrada sea distinta de cero.

12.8. Especificaciones del equipo

A continuación se enumerarán todas las especificaciones generales, eléctricas, térmicas y técnicas del equipo.

12.8.1. Especificaciones generales

1. Funcionamiento autónomo.
2. Simulación de funciones de transferencia SISO continuas y discretas.
3. Configuración de los parámetros principales de la función de transferencia.
4. Configuración de un ruido de la señal de salida.
5. Interfaz gráfica para visualización de la simulación.
6. Posibilidad de alimentación USB (tipo A-B).
7. Display LCD de 4 líneas y 20 caracteres.
8. Teclado matricial 4x4 de membrana.

12.8.2. Especificacione eléctricas y térmicas del equipo

1. Alimentación: Red eléctrica 230V, 50Hz.
2. Consumo medio aproximado: 3w.
3. Señal de salida en el rango de 0-5vdc.
4. Señal de entrada en el rango de 0-5vdc.
5. Rango de temperatura para su funcionamiento: $0^{\circ}C - 50^{\circ}C$

12.8.3. Especificaciones técnicas

1. Reloj interno a 16MHz.
2. Periodo de muestreo configurable de 10ms a 1s.
3. Margen de error del periodo de muestreo menor del 1 %.
4. Funciones de transferencia continuas aproximadas por Euler ó por Tustin.
5. Dos posibilidades de entrada: PWM a 32kHz (0-5vdc) ó continua (0-5vdc).
6. Punto de prueba del periodo de muestreo en el pin digital 2 del Arduino.
7. Diversas posibilidades de ruido de la salida: Sin ruido, ruido del 0.005 %, 0.01 % y 0.1 %.

Capítulo 13

Anexo 5: Código fuente Arduino

En este anexo se presenta todo el código fuente que debe ser grabado en el Arduino Mega 2560. El código ha sido programado en el lenguaje Arduino con la ayuda de librerías para el teclado, la transmisión por I2C y para la comunicación con el display LCD.

13.1. Librerías y variables

```
1  // Librerías utilizadas
2  #include <Keypad.h>
3  #include <Wire.h>
4  #include <LiquidCrystal_I2C.h>
5
6  //////////////// Variables globales ////////////////
7  #define Salida_Pwm 9
8  int i=0,j=0;            // i=> Muestra actual. j=> Contador auxiliar.
9  int ruido=0;            // Valor inicial de ruido de la salida: 0.
10 const int n=5;          // Numero de valores simulados guardados.
11 float x[n],y[n],uu=0; // x=>Entrada, y=>Salida, uu==>Generación de senos.
12 int tipo;              // Tipos de funciones de transferencia según orden
13                        // y método de aproximación seleccionado.
14 //Parámetros de funciones de transferencia reales.
15 float a3=0,a2=0,a1=0,a0=0,b3=0,b2=0,b1=0,b0=0;
16 float coef[8]={0, 0, 0, 0, 0, 0, 0, 0};  // Guardado previo de ellos.
17 //Parámetros de la ecuación en diferencias
18 float k1,k2,k3,k4,k5,k6,k7,T=0.01;       // Periodo=10ms.
19 // Variables de control del menú
20   int menu=0;                          // Control menu
21   int contador=0,func=0;               // Control submenu
22   boolean metodo = false;              // Métodos de apoximación
23   boolean flag=false;                  // Puerto serie desactivado
```

```
24   int l=0,m=1;                          // Variables auxiliares
25   char num_texto[1];                    // Parámetro actual en texto
26   char numero[]="0000000";              // Guardado de números introducidos
27   long numAleatorio;                    // Entero aleatorio: ruido
28   float Aleatorio_float;                // Salida del sistema con ruido
29 //////////////////////////////////////////////
30 \end{lstlisting}
31 \section{Configuración del LCD y Teclado}
32 \label{Ardu2}
33 \begin{lstlisting}[language=C]
34 //////////////////(CONFIGUACIÓN//////////////////
35 //TECLADO
36   const byte ROWS = 4; // Cuatro filas
37   const byte COLS = 4; // Cuatro columnas
38   char keys[ROWS][COLS] = // Distribución de teclas
39   {
40     {'1','2','3','A'},
41     {'4','5','6','B'},
42     {'7','8','9','C'},
43     {'*','0','#','D'}
44   };
45   byte rowPins[ROWS] = {39, 41, 43, 45}; // Filas en estos pines
46   byte colPins[COLS] = {47, 49, 51, 53}; // Columnas en estos pines
47
48   Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
49   char key;       // Tecla pulsada
50 //FIN TECLADO
51
52 // LCD
53 LiquidCrystal_I2C lcd(0x27,20,4);     // 0x27=>i2c. 20 caracteres 4 filas.
54 // FIN LCD
```

13.2. Configuración de la simulación: función *setup*

```
1 void setup()
2 {
3   // Genera semilla aleatoria. Entrada analógica 10 no conectada.
4   randomSeed(analogRead(A10));
5   // Salida de la función de transferencia (PWM).
6   pinMode (Salida_Pwm, OUTPUT);
7   // Punto de prueba para comprobar el periodo de muestreo
8   pinMode (2,OUTPUT);
9   digitalWrite(2,LOW);
```

```
10  // Comienza la salida del sistema al rango máximo.
11  analogWrite (Salida_Pwm,0);
12  //Configuración de la frecuencia de la señal PWM
13  int mascara = 7;
14  int Prescaler = 1;
15  TCCR2B &= ~mascara;
16  TCCR2B |= Prescaler;      // Frecuencia de:31372.55 Hz pin 9.
17  Serial.begin(115200);     //Configuración del puerto serie.
18
19  // Inicio de la configuración de la simulación.
20  // Interactuar con display y teclado.
21  // En este apartado se muestran todos las pantallas que
22  // irán apareciendo en el LCD a medida que el usuario
23  // entre en cada uno de los menús. Se usan funciones
24  // auxiliares donde se realizan las operaciones necesarias
25  // para guardar y tratar la información introducida por
26  // el usuario y por medio del teclado.
27  while (menu!=5)
28  {
29    switch (menu)
30    {
31      //////////////////////////////////////////////////////////////
32      //////////////////////////////////////////////////////////////
33      case 0:
34        inicio_programa_lcd();          // Pantalla principal del programa
35        while (menu==0)
36        {
37          key = keypad.getKey();
38          if (key=='#') menu=1;         // #==> Iniciar configuración fdt
39          else if (key=='*') menu=10; // *==> Configuración de simulación.
40        }
41        break;
42      case 10:
43        pantalla_config();            // Pantalla de configuración.
44        while (menu==10)
45        {
46          key = keypad.getKey();
47          if (key=='1')               // Pantalla del puerto serie.
48          {
49            lcd.clear();
50            delay(20);
51            lcd.print(" Configuracion del  ");
52            lcd.setCursor(0,1);
53            lcd.print("    puerto serie:   ");
54            lcd.setCursor(0,2);
```

```
55              lcd.print("1)Activar          ");
56              lcd.setCursor(0,3);
57              lcd.print("2)Desactivar        ");
58              menu=20;
59              while (menu==20)
60              {
61                key = keypad.getKey();
62                if (key=='1')              // Activar puerto serie.
63                {
64                  lcd.clear();
65                  delay(20);
66                  lcd.print("Envio de datos de la");
67                  lcd.setCursor(0,1);
68                  lcd.print(" simulacion por el  ");
69                  lcd.setCursor(0,2);
70                  lcd.print("puerto serie ha sido");
71                  lcd.setCursor(0,3);
72                  lcd.print("      ACTIVADO      ");
73                  flag=true;
74                  delay(1000);
75                  pantalla_config();
76                  menu=10;
77                }
78                else if (key=='2')       // Desactivar puerto serie.
79                {
80                  lcd.clear();
81                  delay(20);
82                  lcd.print("Envio de datos de la");
83                  lcd.setCursor(0,1);
84                  lcd.print(" simulacion por el  ");
85                  lcd.setCursor(0,2);
86                  lcd.print("puerto serie ha sido");
87                  lcd.setCursor(0,3);
88                  lcd.print("    DESACTIVADO     ");
89                  flag=false;
90                  delay(1000);
91                  pantalla_config();
92                  menu=10;
93                }
94                else if (key=='A')       // Ir al menú anterior.
95                {
96                  pantalla_config();
97                  menu=10;
98                }
99              }
```

```
100           }
101           else if (key=='2')      // Configuración del ruido de salida.
102           {
103             lcd.clear();
104             delay(20);
105             lcd.print("1)Sin ruido          ");
106             lcd.setCursor(0,1);
107             lcd.print("2)Ruido del 0.005%  ");
108             lcd.setCursor(0,2);
109             lcd.print("3)Ruido del 0.01%   ");
110             lcd.setCursor(0,3);
111             lcd.print("4)Ruido del 0.1%    ");
112             menu=30;
113             while (menu==30)
114             {
115               key = keypad.getKey();
116               if (key=='1')           // Señal de salida sin ruido.
117               {
118                 lcd.clear();
119                 delay(20);
120                 lcd.setCursor(0,1);
121                 lcd.print("Simulacion de salida");
122                 lcd.setCursor(0,2);
123                 lcd.print("     SIN RUIDO      ");
124                 ruido=0;
125                 delay(1000);
126                 pantalla_config();
127                 menu=10;
128               }
129               if (key=='2')           // Ruido del 0.005%
130               {
131                 lcd.clear();
132                 delay(20);
133                 lcd.setCursor(0,1);
134                 lcd.print("Simulacion de salida");
135                 lcd.setCursor(0,2);
136                 lcd.print("con ruido del 0.005%");
137                 ruido=5;
138                 delay(1000);
139                 pantalla_config();
140                 menu=10;
141               }
142               if(key=='3')            // Ruido del 0.01%
143               {
144                 lcd.clear();
```

```
145                 delay(20);
146                 lcd.setCursor(0,1);
147                 lcd.print("Simulacion de salida");
148                 lcd.setCursor(0,2);
149                 lcd.print("con ruido del 0.01% ");
150                 ruido=10;
151                 delay(1000);
152                 pantalla_config();
153                 menu=10;
154               }
155               if (key=='4')        // Ruido del 0.1%
156               {
157                 lcd.clear();
158                 delay(20);
159                 lcd.setCursor(0,1);
160                 lcd.print("Simulacion de salida");
161                 lcd.setCursor(0,2);
162                 lcd.print("con ruido del 0.1%  ");
163                 ruido=100;
164                 delay(1000);
165                 pantalla_config();
166                 menu=10;
167               }
168               else if (key=='A')
169               {
170                 pantalla_config();    // Menú anterior
171                 menu=10;
172               }
173             }
174           }
175           else if (key=='A')          // Menú principal
176           {
177             menu=0;
178           }
179         }
180         break;
181       ////////////////////////////////////////////////////////////////
182       ////////////////////////////////////////////////////////////////
183       case 1:    // Menú de elección del tipo de función de transferencia.
184         lcd.clear();
185         delay(20);
186         lcd.print("Tipo de la funcion  ");
187         lcd.setCursor(0,1);
188         lcd.print("de transferencia:   ");
189         lcd.setCursor(0,2);
```

```
190         lcd.print("-->Funcion discreta ");
191         lcd.setCursor(0,3);
192         lcd.print("   Funcion continua ");
193         lcd.noBlink();
194         lcd.noCursor();
195         while (menu==1)
196         {
197           key = keypad.getKey();
198           if (key=='A')          // Ir al menú anterior.
199           {
200             menu=0;
201             contador=0;
202           }
203           Eleccion_func(key);
204         }
205         break;
206       ////////////////////////////////////////////////////////////////
207       ////////////////////////////////////////////////////////////////
208       case 2:          // Pantalla de selección del tipo de aproximación.
209         contador=0;
210         switch (func)
211         {
212           case 1:
213             grado_discreta();
214             break;
215           case 2:
216             lcd.clear();
217             delay(20);
218             lcd.print("Metodo aproximacion ");
219             lcd.setCursor(0,1);
220             lcd.print("funciones continuas:");
221             lcd.setCursor(0,2);
222             lcd.print("1)Euler hacia atras ");
223             lcd.setCursor(0,3);
224             lcd.print("2)Metodo Tustin     ");
225             lcd.noBlink();
226             lcd.noCursor();
227             menu=5;
228             key='0';
229             metodo_aprox();          //Selección del metodo de aprox.
230             lcd.clear();
231             lcd.print(" Funcion continua   ");
232             lcd.setCursor(0,1);
233             lcd.print("Eliga el grado de la");
234             lcd.setCursor(0,2);
```

```
235             lcd.print("funcion de transf.: ");
236             lcd.setCursor(0,3);
237             lcd.print("Grado 1, 2 o 3:");
238             lcd.cursor();
239             lcd.blink();
240             grado_continua();
241             break;
242           }
243         ////////////////////////////////////////////////////////////////
244         ////////////////////////////////////////////////////////////////
245         case 3:         // Pantalla de introducción de parámetros de la fdt.
246           lcd.clear();
247           lcd.print("Parametros de la    ");
248           lcd.setCursor(0,1);
249           lcd.print("funcion de transf.  ");
250           lcd.setCursor(0,2);
251           lcd.print("Numerador:          ");
252           while (menu==3)
253           {
254             parametros_grado();
255             menu=4;
256           }
257           break;
258         ////////////////////////////////////////////////////////////////
259         ////////////////////////////////////////////////////////////////
260         case 4:    // Pantalla mostrada mientras se simula la fdt.
261           contador=0;
262           lcd.clear();
263           delay(20);
264           lcd.print("Simulando la funcion");
265           lcd.setCursor(0,1);
266           lcd.print("de transferencia... ");
267           if (flag==true)          // Indica si el puerto serie está activado
268           {
269             lcd.setCursor(0,2);
270             lcd.print("Los datos se envian ");
271             lcd.setCursor(0,3);
272             lcd.print("por el puerto serie ");
273           }
274           else
275           {
276             lcd.setCursor(0,2);
277             lcd.print("    Puerto serie    ");
278             lcd.setCursor(0,3);
279             lcd.print("    DESACTIVADO     ");
```

```
280           }
281           lcd.noBlink();
282           lcd.noCursor();
283           param_ec_dif();
284           config_timer();
285           menu=5;
286           break;
287         }
288     }
289 }
290 ////////////////////////////////////////////////////////////////////////
291 ////////////////////////////////////////////////////////////////////////
```

13.3. Función *loop*

```
1  ////////////////////////////////////////////////////////////////////////
2  ////////////////////////////////////////////////////////////////////////
3  // Programa principal
4  void loop()
5  {
6    key = keypad.getKey();      // Reinicio en caso de pulsar #.
7    if (key=='#') reinicio();
8  }
9  ////////////////////////////////////////////////////////////////////////
10 ////////////////////////////////////////////////////////////////////////
```

13.4. Subrutina de vectorización de overflow del *timer1*

```
1  ////////////////////////////////////////////////////////////////////////
2  ////////////////////////////////////////////////////////////////////////
3  //Subrutina de vectorización de overflow del TIMER1
4  ISR(TIMER1_OVF_vect)
5  {
6    // Recarga el valor del temporizador
7    // En T se introduce el periodo de muestreo deseado (entre 10ms y 1s).
8    TCNT1=(1.048576-T+0.00005)/16e-6;     // 0.00005 => Corrección del periodo
9                                          // de muestreo.
10
11   //Señal senoidal de simulación
12   //uu=uu+0.017;
13   //x[i]=sin(uu);
```

```
14
15    //Señal escalón de simulación
16    //x[i]=1;
17
18    // Señal analógica externa.
19    x[i]=analogRead(0);
20    x[i]=x[i]*100/1023;
21
22    //Primera muestra nula siempre.
23    x[0]=0;
24
25    if (tipo==1 or tipo==4 or tipo==7)     // Fdt de grado 1.
26    {                                      // Ecuación en diferencias
27      if ((i-1)<0)  {k2=k3=0;}
28      y[i]=k1*x[i]+k2*x[i-1]-k3*y[i-1];
29      if ((i-1)<0)  {param_ec_dif();}
30    }
31    else if (tipo==2 or tipo==5 or tipo==8)  // Fdt de grado 2.
32    {                                        // Ecuación en diferencias
33      if ((i-1)<0)  {k2=k3=k4=k5=0;}
34      else if ((i-2)<0) {k3=k5=0;}
35      y[i]=k1*x[i]+k2*x[i-1]+k3*x[i-2]-k4*y[i-1]-k5*y[i-2];
36      if ((i-2)<0) {param_ec_dif();}
37    }
38    else if (tipo==3 or tipo==6 or tipo==9)  // Fdt de grado 3.
39    {                                        // Ecuación en diferencias
40      if ((i-1)<0)  {k2=k3=k4=k5=k6=k7=0;}
41      else if ((i-2)<0) {k3=k4=k6=k7=0;}
42      else if ((i-3)<0) {k4=k7=0;}
43      y[i]=k1*x[i]+k2*x[i-1]+k3*x[i-2]+k4*x[i-3]-k5*y[i-1]-k6*y[i-2]-k7*y[i
            -3];
44      if ((i-3)<0) {param_ec_dif();}
45    }
46    // Generación del ruido
47    Genera_ruido();
48    y[i]=Aleatorio_float;
49    // Simulación
50    if (tipo!=0 && menu==5)
51    {
52      if (i<(n-1))                  // Primeras 4 muestras.
53      {
54        if (flag==true)              // Puerto serie.
55        {
56          Serial.println(y[i],4);
57          Serial.println(x[i]);
```

```
58        }
59        analogWrite (Salida_Pwm,y[i]*255/100); // Valor actual de salida.
60        i++;
61      }
62      else                          // Resto de la simulación (INFINITO).
63      {
64        if (flag==true)              // Puerto serie
65        {
66          Serial.println(y[i],4);
67          Serial.println(x[i]);
68        }
69        analogWrite (Salida_Pwm,y[i]*255/100); // Valor actual de salida.
70        for(j=0;j<n-1;j++)          // Desplaza valores una posición a izq.
71        {                           // para calcular la salida siguiente.
72          y[j]=y[j+1];
73          x[j]=x[j+1];
74        }
75      }
76    }
77    else i=0;
78    digitalWrite(2,!digitalRead(2));
79  }
80  ////////////////////////////////////////////////////////////////////////////
81  ////////////////////////////////////////////////////////////////////////////
```

13.5. Resto de funciones utilizadas

```
1
2  ////////////////////////////////////////////////////////////////////////////
3  ////////////////////////////////////////////////////////////////////////////
4  // FUNCIONES EMPLEADAS.
5  // Pantalla de inicio del programa
6  void inicio_programa_lcd()
7  {
8    lcd.init();
9    lcd.backlight();
10   lcd.print("Emulador de funcion ");
11   lcd.setCursor(0,1);
12   lcd.print("  de transferencia  ");
13   lcd.setCursor(0,2);
14   lcd.print("      S.I.S.O.      ");
15   lcd.setCursor(0,3);
16   lcd.print("Pulse # para empezar");
```

```
17    lcd.noCursor();
18    }
19
20  // Pantalla de configuración (al pulsar *)
21  void pantalla_config ()
22  {
23    lcd.clear();
24    delay(20);
25    lcd.print("Configuracion de la ");
26    lcd.setCursor(0,1);
27    lcd.print("     simulacion:    ");
28    lcd.setCursor(0,2);
29    lcd.print("1)Puerto serie     ");
30    lcd.setCursor(0,3);
31    lcd.print("2)Ruido de la salida");
32  }
33
34  // Para la simulación y reinicia el programa (al pulsar #).
35  void reinicio ()
36    {
37      TIMSK1=(0<<TOIE1);          // Para el timer 1.
38      menu=0;                     // Vuelve al menú principal.
39      tipo=0;
40      for (int ind=0;ind<n;ind++)   // Inicializa entrada y salida.
41      {
42        y[ind]=0;
43        x[ind]=0;
44      }
45      i=0;                        // Reset instante de muestreo
46      setup();                    // Inicio de la secuencia de configuración.
47      key='0';
48    }
49
50  // Desplazamiento por el menú de diversas funciones de transferencia.
51  void despl_menu(int cont)
52  {
53    switch (cont)
54    {
55      case 0:
56        lcd.setCursor(0,2);
57        lcd.print("-->Funcion discreta ");
58        lcd.setCursor(0,3);
59        lcd.print("   Funcion continua ");
60        break;
61      case 1:
```

```
62        lcd.setCursor(0,2);
63        lcd.print("   Funcion discreta ");
64        lcd.setCursor(0,3);
65        lcd.print("-->Funcion continua ");
66        break;
67      case 2:
68        lcd.setCursor(0,2);
69        lcd.print("   Funcion continua ");
70        lcd.setCursor(0,3);
71        lcd.print("-->Motor de CC      ");
72        break;
73      case 3:
74        lcd.setCursor(0,2);
75        lcd.print("   Motor de CC      ");
76        lcd.setCursor(0,3);
77        lcd.print("-->Circuito RLC     ");
78        break;
79      case 4:
80        lcd.setCursor(0,2);
81        lcd.print("   Circuito RLC     ");
82        lcd.setCursor(0,3);
83        lcd.print("-->Masa oscilante   ");
84        break;
85      case 5:
86        lcd.setCursor(0,2);
87        lcd.print("   Masa oscilante   ");
88        lcd.setCursor(0,3);
89        lcd.print("-->Control Temp.    ");
90        break;
91      case 6:
92        lcd.setCursor(0,2);
93        lcd.print("   Control Temp.    ");
94        lcd.setCursor(0,3);
95        lcd.print("-->Tanques acoplados");
96        break;
97      case 7:
98        lcd.setCursor(0,2);
99        lcd.print("   Tanques acoplados");
100       lcd.setCursor(0,3);
101       lcd.print("-->Aeropendulo      ");
102       break;
103     case 8:
104       lcd.setCursor(0,2);
105       lcd.print("   Aeropendulo      ");
106       lcd.setCursor(0,3);
```

```
107       lcd.print("-->Amort.-Carro-Res.");
108       break;
109     case 9:
110       lcd.setCursor(0,2);
111       lcd.print("   Amort.-Carro-Res.");
112       lcd.setCursor(0,3);
113       lcd.print("-->Nivel Deposito   ");
114       break;
115     case 10:
116       lcd.setCursor(0,2);
117       lcd.print("   Nivel Deposito   ");
118       lcd.setCursor(0,3);
119       lcd.print("-->Temp. de proceso ");
120       break;
121     case 11:
122       lcd.setCursor(0,2);
123       lcd.print("   Temp. de proceso ");
124       lcd.setCursor(0,3);
125       lcd.print("-->Sistema de mezcla");
126       break;
127     case 12:
128       lcd.setCursor(0,2);
129       lcd.print("   Sistema de mezcla");
130       lcd.setCursor(0,3);
131       lcd.print("-->Posic. Antena    ");
132       break;
133     case 13:
134       lcd.setCursor(0,2);
135       lcd.print("   Posic. Antena    ");
136       lcd.setCursor(0,3);
137       lcd.print("-->Posic. Camaras   ");
138       break;
139     case 14:
140       lcd.setCursor(0,2);
141       lcd.print("   Posic. Camaras   ");
142       lcd.setCursor(0,3);
143       lcd.print("-->Posic. Angular   ");
144       break;
145     case 15:
146       lcd.setCursor(0,2);
147       lcd.print("   Posic. Camaras   ");
148       lcd.setCursor(0,3);
149       lcd.print("-->Movimiento barco ");
150       break;
151     case 16:
```

```
152       lcd.setCursor(0,2);
153       lcd.print("   Movimiento barco ");
154       lcd.setCursor(0,3);
155       if (flag==true) lcd.print("-->DES. Puerto serie");
156       else lcd.print("-->ACT. Puerto serie");
157       break;
158    }
159 }
160
161 // Control del desplazamiento por el menú anterior.
162 void Eleccion_func(char tecla)
163 {
164    if (tecla=='C')              // Retrocede una posición.
165    {
166      if (contador>0)
167      {
168        contador=contador-1;
169        despl_menu(contador);
170      }
171      else
172      {
173        contador=0;
174      }
175    }
176    else if (tecla=='D')         // Avanza una posición.
177    {
178      if (contador<15)
179      {
180        contador=contador+1;
181        despl_menu(contador);
182      }
183      else
184      {
185        contador=15;
186      }
187    }
188    else if (tecla=='#')         // Selecciona la función de
189    {                            // transferencia actual.
190      if (contador==0)           // Función discreta seleccionada.
191      {
192        lcd.clear();
193        lcd.print("  Funcion discreta  ");
194        lcd.setCursor(0,1);
195        lcd.print("Eliga el grado de la");
196        lcd.setCursor(0,2);
```

```
197       lcd.print("funcion de transf.: ");
198       lcd.setCursor(0,3);
199       lcd.print("Grado 1, 2 o 3:");
200       lcd.blink();
201       func=1;
202       menu=2;
203     }
204     else if (contador==1)  // Funcion continua seleccionada.
205     {
206       lcd.clear();
207       func=2;
208       menu=2;
209     }
210     else if (contador==2)  // Funcion motor de CC.
211     {
212       lcd.clear();
213       lcd.print("    Motor de CC.    ");
214       delay(500);
215       tipo=5;
216       a3=a2=a1=0;
217       a0=0.01;
218       b3=0;
219       b2=0.005;
220       b1=0.06;
221       b0=0.1001;
222       func=4;
223       menu=4;
224     }
225     else if (contador==3)  // Función circuito RLC
226     {
227       lcd.clear();
228       lcd.print("    Circuito RLC    ");
229       delay(500);
230       tipo=5;
231       a3=a2=a1=0;
232       a0=17;
233       b3=0;
234       b2=1;
235       b1=8;
236       b0=17;
237       func=4;
238       menu=4;
239     }
240     else if (contador==4)  // Función Masa oscilante
241     {
```

```
242       lcd.clear();
243       lcd.print("   Masa oscilante   ");
244       delay(500);
245       tipo=5;
246       a3=a2=a1=0;
247       a0=0.0192;
248       b3=0;
249       b2=1;
250       b1=2.353;
251       b0=3.84;
252       func=4;
253       menu=4;
254     }
255     else if (contador==5)  // Función control de temperatura.
256     {
257       lcd.clear();
258       lcd.print("    Control Temp.   ");
259       delay(500);
260       tipo=5;
261       a3=a2=a1=0;
262       a0=0.001;
263       b3=0;
264       b2=1;
265       b1=0.041;
266       b0=0.0012;
267       func=4;
268       menu=4;
269     }
270     else if (contador==6)  // Función tanques acoplados.
271     {
272       lcd.clear();
273       lcd.print(" Tanques acoplados  ");
274       delay(500);
275       tipo=5;
276       a3=a2=a1=0;
277       a0=1.5;
278       b3=0;
279       b2=1800;
280       b1=85;
281       b0=1;
282       func=4;
283       menu=4;
284     }
285     else if (contador==7)  // Función del aeropéndulo
286     {
```

```
287         lcd.clear();
288         lcd.print("     Aeropendulo      ");
289         delay(500);
290         tipo=8;
291         a3=a2=a0=0;
292         a1=0.005487;
293         b3=0;
294         b2=1;
295         b1=-0.5666;
296         b0=-0.4214;
297         func=4;
298         menu=4;
299       }
300       else if (contador==8)  // Función sistema compuesto.
301       {
302         lcd.clear();
303         lcd.print("Sistema compuesto:  ");
304         lcd.setCursor(0,1);
305         lcd.print("Amortiguador-Carro- ");
306         lcd.setCursor(0,2);
307         lcd.print("-Resorte            ");
308         delay(500);
309         tipo=6;
310         a3=a2=a1=0;
311         a0=1;
312         b3=1;
313         b2=4;
314         b1=5;
315         b0=2;
316         func=4;
317         menu=4;
318       }
319       else if (contador==9)  // Función nivel depósito
320       {
321         //Aproximación del retardo con Matlab
322         lcd.clear();
323         lcd.print("  Nivel Deposito    ");
324         delay(500);
325         tipo=6;
326         a3=0;
327         a2=1.5;
328         a1=-45;
329         a0=450;
330         b3=40;
331         b2=1201;
```

```
332         b1=12030;
333         b0=300;
334         func=4;
335         menu=4;
336 
337       }
338       else if (contador==10)  // Función control temperatura
339       {
340         //Aproximación del retardo con Matlab
341         lcd.clear();
342         lcd.print("Control temperatura ");
343         lcd.setCursor(0,1);
344         lcd.print("   de un proceso    ");
345         delay(500);
346         tipo=6;
347         a3=0;
348         a2=0;
349         a1=-5000;
350         a0=100000;
351         b3=1;
352         b2=80;
353         b1=1700;
354         b0=10000;
355         func=4;
356         menu=4;
357       }
358       else if (contador==11)  // Función sistema de mezclado
359       {
360         //Aproximación del retardo con Matlab
361         lcd.clear();
362         lcd.print("Sistema de mezclado ");
363         delay(500);
364         tipo=6;
365         a3=0;
366         a2=1;
367         a1=-4.286;
368         a0=6.122;
369         b3=15;
370         b2=66.29;
371         b1=100.4;
372         b0=12.24;
373         func=4;
374         menu=4;
375       }
376       else if (contador==12)  // Función posicionamiento antena
```

```
377     {
378       lcd.clear();
379       lcd.print("Posicionamiento de  ");
380       lcd.setCursor(0,1);
381       lcd.print("antena parabólica   ");
382       delay(500);
383       tipo=6;
384       a3=a2=a1=0;
385       a0=1;
386       b3=2;
387       b2=3;
388       b1=2;
389       b0=0;
390       func=4;
391       menu=4;
392     }
393     else if (contador==13)  // Función posicionamiento camaras
394     {
395       lcd.clear();
396       lcd.print("Posicionamiento de  ");
397       lcd.setCursor(0,1);
398       lcd.print("camaras de escenario");
399       delay(500);
400       tipo=6;
401       a3=a2=a1=0;
402       a0=123632.8;
403       b3=1;
404       b2=87.6;
405       b1=5329;
406       b0=0;
407       func=4;
408       menu=4;
409     }
410     else if (contador==14)  // Funcion posicionamiento angular
411     {
412       lcd.clear();
413       lcd.print("Posicionamiento     ");
414       lcd.setCursor(0,1);
415       lcd.print("angular de una carga");
416       delay(500);
417       tipo=6;
418       a3=a2=a1=0;
419       a0=12968.469;
420       b3=1;
421       b2=666.911;
```

```
422         b1=288.315;
423         b0=0;
424         func=4;
425         menu=4;
426       }
427       else if (contador==15)  // Funcion de movimiento de un barco
428       {
429         lcd.clear();
430         lcd.print("    Sistema de      ");
431         lcd.setCursor(0,1);
432         lcd.print("    movimiento      ");
433         lcd.setCursor(0,2);
434         lcd.print("    de un barco     ");
435         delay(500);
436         tipo=5;
437         a3=a2=a1=0;
438         a0=0.0000828157;
439         b3=0;
440         b2=1;
441         b1=0.00472;
442         b0=0;
443         func=4;
444         menu=4;
445       }
446     }
447 }
448 
449 // Grado y coeficientes de la funcion de transferencia.
450 void parametros_grado()
451 {
452   pide_grado();
453   a3=coef[4];
454   a2=coef[3];
455   a1=coef[2];
456   a0=coef[1];
457   lcd.clear();
458   lcd.print("Parametros de la    ");
459   lcd.setCursor(0,1);
460   lcd.print("funcion de transf.  ");
461   lcd.setCursor(0,2);
462   lcd.print("Denominador:        ");
463   pide_grado();
464   b3=coef[4];
465   b2=coef[3];
466   b1=coef[2];
```

```
467   b0=coef[1];
468   lcd.clear();
469   if (tipo>6)
470   {
471     lcd.print("Parametros de la    ");
472     lcd.setCursor(0,1);
473     lcd.print("funcion de transf.  ");
474     lcd.setCursor(0,2);
475     lcd.print("Periodo de muestreo:");
476     lcd.setCursor(0,3);
477     lcd.print("T (0.01s a 1s):");
478     l=1;
479     key='0';
480     coeficientes();
481     T=coef[1];
482     if (T<0.01) T=0.01;
483     if (T>1) T=1;
484   }
485   else T=0.01; //Periodo de muestreo mínimo para simular sistemas continuos
486 }
487
488 // Función para pedir los coeficientes de las funciones de transferencia.
489 // Se utiliza para los coeficientes del numerador y del denominador.
490 void pide_grado ()
491 {
492   if (tipo==1 or tipo==4 or tipo==7) l=2;  // Grado 1, dos coeficientes
493   if (tipo==2 or tipo==5 or tipo==8) l=3;  // Grado 2, tres coeficientes
494   if (tipo==3 or tipo==6 or tipo==9) l=4;  // Grado 3, cuatro coeficientes
495   while (l!=0)
496   {
497     lcd.setCursor(0,3);
498     lcd.print("Grado ");
499     lcd.print(itoa(l-1,num_texto,10));     // Pasa a array de char.
500     lcd.write(':');
501     lcd.print("            ");
502     lcd.setCursor(8,3);
503     key='0';
504     coeficientes();
505   }
506 }
507
508 // Subrutina que se encarga de obtener los coeficientes introducidos
509 // por el teclado. Estos pueden ser negativos y pueden tener un punto
510 // decimal. Tambien se utiliza para recibir el periodo de muestreo
511 // introducido por el usuario.
```

```
512 void coeficientes ()
513 {
514   boolean flag=false;
515   boolean coma=false;
516   int indice_coma=0;
517   m=0;
518   while(key!='#')
519     {
520       key = keypad.getKey();
521       if (key)
522       {
523         if (key=='C' && !flag && m==0)        // Al pulsar C, signo negativo
524         {
525           key='-';
526           flag=true;
527           lcd.write(key);
528         }
529         else if (key=='*' && !coma && m>0 && m<6)  // [*]=> punto decimal
530         {
531           key='.';
532           coma=true;
533           indice_coma=m;
534           lcd.write(key);
535           numero[m]=key;
536           m++;
537         }
538         else if (key=='B')                    // Borra caracter anterior.
539         {
540           if (m==0 && flag==true)
541           {
542             flag=false;
543               lcd.command(LCD_CURSORSHIFT);
544               lcd.command(LCD_MOVELEFT);
545               lcd.print(" ");
546               lcd.command(LCD_CURSORSHIFT);
547               lcd.command(LCD_MOVELEFT);
548           }
549           if (m>0)
550           {
551             if (m==indice_coma+1)
552             {
553               coma=false;
554               indice_coma=0;
555             }
556               lcd.command(LCD_CURSORSHIFT);
```

```
557                 lcd.command(LCD_MOVELEFT);
558                 lcd.print(" ");
559                 lcd.command(LCD_CURSORSHIFT);
560                 lcd.command(LCD_MOVELEFT);
561               m--;
562               numero[m]=0;
563             }
564           }
565           else if (key!='A' && key!='B'&& key!='C' && key!='D' && key!='*'
                  && key!='#' && m<7)
566           {
567             lcd.write(key);  // Sólo se escriben los números pulsados.
568             numero[m]=key;
569             m++;
570           }
571       }
572     }
573     numero[m]='.';                    // Flag de fin.
574     m++;                              // Siguiente cifra.
575     for (;m<7;m++) numero[m]='0';     // Completa con ceros el número
576     if (flag) coef[l]=-atof(numero); // Convierte el array de chars
577     else coef[l]=atof(numero);        // en un float con el signo
578     for (int ij=0;ij<7;ij++)          // Reinicia el array de chars a cero.
579     {
580       numero[ij]='0';
581     }
582     l--;                             // Coeficiente introducido.
583 }
584
585 // Grado de funciones de transferencia discretas
586 void grado_discreta()
587 {
588   while(menu==2)
589   {
590     key = keypad.getKey();
591     switch (key)
592     {
593       case '1':           // Grado 1.
594         lcd.write(key);
595         delay(300);
596         tipo=7;
597         menu=3;
598         break;
599       case '2':           // Grado 2.
600         lcd.write(key);
```

```
601           delay(300);
602           tipo=8;
603           menu=3;
604           break;
605         case '3':           // Grado 3.
606           lcd.write(key);
607           delay(300);
608           tipo=9;
609           menu=3;
610           break;
611         case 'A':           // Regresa al menú anterior.
612           menu=1;
613           break;
614       }
615     }
616 }
617
618 // Elección del método de aproximación funciones continuas
619 void metodo_aprox()
620 {
621   while (menu==5)
622   {
623     key = keypad.getKey();
624     if (key=='1')           // Método de Euler.
625     {
626       lcd.clear();
627       lcd.print("Ha seleccionado el  ");
628       lcd.setCursor(0,1);
629       lcd.print("metodo de Euler     ");
630       delay(600);
631       metodo=true;
632       menu=2;
633     }
634     else if (key=='2')      // Método de Tustin.
635     {
636       lcd.clear();
637       lcd.print("Ha seleccionado el  ");
638       lcd.setCursor(0,1);
639       lcd.print("metodo de Tustin    ");
640       delay(600);
641       menu=2;
642       metodo=false;
643     }
644     else if (key=='A') menu=1;  // Menú anterior.
645   }
```

```
646 }
647
648 // Pide el grado de funciones de transferencia continuas
649 void grado_continua()
650 {
651   while (menu==2 && key!='A')
652   {
653     key = keypad.getKey();
654     switch (key)
655     {
656       case '1':           // Grado 1.
657         lcd.write(key);
658         delay(300);
659         if (metodo)      // Tipo en función del método aprox.
660         {
661           tipo=1;
662           menu=3;
663         }
664         else
665         {
666           tipo=4;
667           menu=3;
668         }
669         break;
670       case '2':           // Grado 2.
671         lcd.write(key);
672         delay(300);
673         if (metodo)       // Tipo en función del método aprox.
674         {
675           tipo=2;
676           menu=3;
677         }
678         else
679         {
680           tipo=5;
681           menu=3;
682         }
683         break;
684       case '3':           // Grado 3.
685         lcd.write(key);
686         delay(300);
687         if (metodo)       // Tipo en función del método aprox.
688         {
689           tipo=3;
690           menu=3;
```

```
691           }
692           else
693           {
694             tipo=6;
695             menu=3;
696           }
697           break;
698         case 'A':           // Menú anterior.
699           menu=2;
700           break;
701       }
702     }
703 }
704
705 // Subrutina de configuración del timer 1
706 // Se encarga de que el periodo de muestreo sea el indicado.
707 // Se utiliza la interrupción por desbordamiento.
708 void config_timer()
709 {
710   cli();                      // Deshabilita interrupciones
711   TCCR1A = 0x00;
712   TCCR1B= 0x00;
713   TCCR1B = (1 << CS12);       //Preescaler de 256
714   TCNT1=(1.048576-T)/16e-6; // Tmax=1,04s; Tmin=16us (Teórico).
715   TIMSK1=(1<<TOIE1);          // Arranca el timer 1.
716   sei();                      // Habilita interrupciones
717 }
718
719 // Esta función calcula los parámetros necesarios para las ecuaciones
720 // en diferencias según el tipo de aproximación seleccionado y el grado
721 // de la función de transferencia introducida.
722 void param_ec_dif()
723 {
724     //Funciones de transferencia continuas de primer orden (EULER)
725   if (tipo==1)
726   {
727     k1 =(a1 + a0*T)/(b1 + b0*T);
728     k2 =-(a1)/(b1 + b0*T);
729     k3 =-(b1)/(b1 + b0*T);
730   }
731   //Funciones de transferencia continuas de segundo orden (EULER)
732   else if (tipo==2)
733   {
734     k1=(a2+a1*T+a0*sq(T))/(b2+b1*T+b0*sq(T));
735     k2=-(2*a2+a1*T)/(b2+b1*T+b0*sq(T));
```

```
736       k3=a2/(b2+b1*T+b0*sq(T));
737       k4=-(2*b2+b1*T)/(b2+b1*T+b0*sq(T));
738       k5=b2/(b2+b1*T+b0*sq(T));
739     }
740     //Funciones de transferencia continuas de tercer orden (EULER)
741     else if (tipo==3)
742     {
743       k1=(a3+a2*T+a1*sq(T)+a0*pow(T,3))/(b3+b2*T+b1*sq(T)+b0*pow(T,3));
744       k2=-(3*a3+2*a2*T+a1*pow(T,2))/(b3+b2*T+b1*sq(T)+b0*pow(T,3));
745       k3=(3*a3+a2*T)/(b3+b2*T+b1*sq(T)+b0*pow(T,3));
746       k4=-(a3)/(b3+b2*T+b1*sq(T)+b0*pow(T,3));
747       k5=-(3*b3+2*b2*T+b1*sq(T))/(b3+b2*T+b1*sq(T)+b0*pow(T,3));
748       k6=(3*b3+b2*T)/(b3+b2*T+b1*sq(T)*b0*pow(T,3));
749       k7=-(b3)/(b3*b2*T+b1*sq(T)+b0*pow(T,3));
750     }
751     //Funciones de transferencia continuas de primer orden (TUSTIN)
752     else if (tipo==4)
753     {
754       k1=(2*a1+a0*T)/(2*b1+b0*T);
755       k2=(a0*T-2*a1)/(2*b1+b0*T);
756       k3=(b0*T-2*b1)/(2*b1+b0*T);
757     }
758     //Funciones de transferencia continuas de segundo orden (TUSTIN)
759     else if (tipo==5)
760     {
761       k1=(4*a2+2*a1*T+a0*sq(T))/(4*b2+2*b1*T+b0*sq(T));
762       k2=(2*a0*sq(T)-8*a2)/(4*b2+2*b1*T+b0*sq(T));
763       k3=(4*a2+a0*sq(T)-2*a1*T)/(4*b2+2*b1*T+b0*sq(T));
764       k4=(2*b0*sq(T)-8*b2)/(4*b2+2*b1*T+b0*sq(T));
765       k5=(4*b2+b0*sq(T)-2*b1*T)/(4*b2+2*b1*T+b0*sq(T));
766     }
767     //Funciones de transferencia continuas de tercer orden (TUSTIN)
768     else if (tipo==6)
769     {
770       k1=(8*a3+4*a2*T+2*a1*sq(T)+a0*pow(T,3))/(8*b3+4*b2*T+2*b1*sq(T)+b0*pow(
              T,3));
771       k2=(3*a0*pow(T,3)+2*a1*sq(T)-4*a2*T-24*a3)/(8*b3+4*b2*T+2*b1*sq(T)+b0*
              pow(T,3));
772       k3=(24*a3-4*a2*T-2*a1*sq(T)+3*a0*pow(T,3))/(8*b3+4*b2*T+2*b1*sq(T)+b0*
              pow(T,3));
773       k4=(a0*pow(T,3)-2*a1*sq(T)+4*a2*T-8*a3)/(8*b3+4*b2*T+2*b1*sq(T)+b0*pow(
              T,3));
774       k5=(3*b0*pow(T,3)+2*b1*sq(T)-4*b2*T-24*b3)/(8*b3+4*b2*T+2*b1*sq(T)+b0*
              pow(T,3));
```

```
775       k6=(24*b3-4*b2*T-2*b1*sq(T)+3*b0*pow(T,3))/(8*b3+4*b2*T+2*b1*sq(T)+b0*
              pow(T,3));
776       k7=(b0*pow(T,3)-2*b1*sq(T)+4*b2*T-8*b3)/(8*b3+4*b2*T+2*b1*sq(T)+b0*pow(
              T,3));
777     }
778     // Funcíón de transferencia discreta de grado 1.
779     else if (tipo==7)
780     {
781       k1=a1/b1;
782       k2=a0/b1;
783       k3=b0/b1;
784     }
785     // Funcíón de transferencia discreta de grado 2.
786     else if (tipo==8)
787     {
788       k1=a2/b2;
789       k2=a1/b2;
790       k3=a0/b2;
791       k4=b1/b2;
792       k5=b0/b2;
793     }
794     // Funcíón de transferencia discreta de grado 3.
795     else if (tipo==9)
796     {
797       k1=a3/b3;
798       k2=a2/b3;
799       k3=a1/b3;
800       k4=a0/b3;
801       k5=b2/b3;
802       k6=b1/b3;
803       k7=b0/b3;
804     }
805 }
806
807 // Subrutina de generación de ruido aleatorio para la salida.
808 void Genera_ruido()
809 {
810   float minimo,maximo;
811   if (ruido==0) Aleatorio_float=y[i];      // Sin ruido, salida teórica.
812   else
813   {
814     if (ruido==5)                          // Ruido del 0.005%.
815     {
816       minimo=999950*y[i];
817       maximo=1000050*y[i];
```

```
818       }
819       else if (ruido==10)                    // Ruido del 0.01%.
820       {
821         minimo=999900*y[i];
822         maximo=1000100*y[i];
823       }
824       else if (ruido==100)                    // Ruido del 0.1%.
825       {
826         minimo=999000*y[i];
827         maximo=1001000*y[i];
828       }
829       numAleatorio = random(minimo,maximo);
830       Aleatorio_float= (float) numAleatorio/1000000;
831     }
832 }
```

Capítulo 14

Anexo 6: Código fuente Processing

En el presente anexo se inclúe el código fuente del *sketch* realizado en Processing para ver los datos de la simulación del equipo. Estos datos estarán disponibles cuando se active el puerto serie del equipo. Para ejecutar el código se debe abrir el archivo con *Processing* cuya interfaz es similar a la del Arduino. Se ejecuta el código siguiendo las instrucciones que aparecen en las líneas 35 a 40 del código siguiente:

14.1. Variables

```
1  // Graphing sketch
2  // Este programa recoge cadenas codificadas en ASCII
3  // desde el puerto serie a 9600 baudios y hace gráficos con ellos. Espera
4  // valores seguidos por un salto de línea.
5
6  import processing.serial.*;
7  Serial myPort;          // El puerto serie
8  int xPos = 0;           // posición horizontal del gráfico
9  float plotX1, plotY1; // Coordenadas del área del gráfico
10 float plotX2, plotY2;
11 int intervalo=25;     // Intervalos indicados en el eje x.
12 PFont plotFont;
13 int row;
14 int rowCount=900;      // Número de puntos ploteados.
15 float dataMin=0;       // Rango de datos de la gráfica.
16 float dataMax=1.001;
17 float intervaloY=0;   // Distancia entre dos intervalos en el eje y.
18 float labelX,labelY;
19 long Xabs=0;           // Cuenta absoluta del número de muestras.
20 int flag=0;
```

14.2. Función *setup*

```
1  void setup () {
2    // Establece el tamaño de la ventana:
3    size(1300, 500);
4    // Número de intervalos marcados en el eje y
5    intervaloY=(dataMax-dataMin)/10;
6
7    // Corners of the plotted time series
8    plotX1 = 100;
9    plotX2 = width - 50;
10   labelX = 35;
11   plotY1 = 60;
12   plotY2 = height - 70;
13   labelY = height - 30;
14
15   // Lista los puertos serie disponibles. Se debe seleccionar
16   // el correspondiente al Arduino (Comprobar puertos COM)
17   println(Serial.list());
18   // Normalmente el puerto serie al que está conectado el arduino
19   // es el primero de la lista: El índice 0: Serial.list()[0].
20   myPort = new Serial(this, Serial.list()[0], 115200);
21   // Genera un serialEvent() cuando reciba un de salto de linea:
22   myPort.bufferUntil('\n');
23   // Establece el fondo inicial:
24   background(224);       // Fondo detrás de la gráfica.
25   fill(0);             // Área de la gráfica en blanco.
26   rectMode(CORNERS);     // Crea la gráfica rectangular.
27   noStroke( );
28   rect(plotX1, plotY1, plotX2, plotY2);
29   strokeWeight(4);
30   stroke(#5679C1);
31   plotFont=createFont("Arial",20);
32   textFont(plotFont);
33   drawAxisLabels( );     // Etiquetas de ambos ejes.
34   drawXLabels( );        // Intervalos del eje x.
35   drawVolumeLabels( );   // Intervalos del eje y.
36   drawTitle();           // Título de la gráfica.
37   smooth( );
38 }
```

14.3. Función draw

```
1 void draw ()
2 {
3   // todo se hace en serialEvent()
4 }
```

14.4. Función *serialEvent*

```
1 void serialEvent (Serial myPort) {
2   // Recoge la cadena ASCII:
3   String inString = myPort.readStringUntil('\n');
4   if (inString != null)
5   {
6     // Elimina cualquier espacio en blanco:
7     inString = trim(inString);
8     // Convierte a entero y mapea al alto de la pantalla:
9     float inByte = float(inString);
10 //    MaxAndMin(inByte);
11 //    if (dataMin>inByte || dataMax<inByte)
12 //    {
13 //      drawVolumeLabels( );
14 //    }
15    //inByte = map(inByte, 0, 2, 0, height);
16    if (flag==0)
17    {
18      stroke(127,34,255);         // Color de línea
19      strokeWeight(6);            // Ancho de línea
20      flag=1;
21    }
22    else
23    {
24      stroke(27,134,0);         // Color de línea
25      strokeWeight(5);          // Ancho de línea
26      flag=0;
27    }
28    float x=map(xPos, 0,rowCount,plotX1,plotX2);
29    float y=map(inByte,0,dataMax,plotY2,plotY1);
30    point(x,y);
31    //line(x, height-plotY1-10, x, y);
32    // en el borde de la pantalla, vuelve al principio:
33    if (xPos >= rowCount)
34    {
```

```
35        xPos = 0;
36        Xabs++;
37        background(224);
38        // Show the plot area as a white box.
39        fill(0);
40        rectMode(CORNERS);
41        noStroke( );
42        rect(plotX1, plotY1, plotX2, plotY2);
43        strokeWeight(4);
44        // Draw the data for the first column.
45        stroke(#5679C1);
46        plotFont=createFont("Arial",20);
47        textFont(plotFont);
48        drawAxisLabels( );
49        drawXLabels( );
50        drawVolumeLabels( );
51        drawTitle();
52        smooth( );
53      }
54      else
55      {
56        // incrementa la posición horizontal:
57        if (flag==0)
58        {
59          xPos++;
60          Xabs++;
61        }
62      }
63    }
64 }
```

14.5. Resto de funciones utilizadas

```
1  void drawXLabels( ) {
2    fill(0);
3    textSize(11);
4    textAlign(CENTER, TOP);
5    stroke(100);
6    strokeWeight(1);
7    for (int row = 0; row <= rowCount; row++)
8    {
9      if (row %intervalo == 0)
10     {
```

```
11        float x = map(row, 0, rowCount, plotX1, plotX2);
12        text(nf(row+Xabs,1,0), x, plotY2 + 5);
13        line(x, plotY1, x, plotY2);
14      }
15    }
16  }
17
18
19  void drawVolumeLabels( )
20  {
21    fill(0);
22    textSize(10);
23    textAlign(RIGHT, CENTER);
24    for (float v = dataMin; v <= dataMax; v += intervaloY)
25    {
26      float y = map(v, dataMin, dataMax, plotY2, plotY1);
27      text(nf(v,1,2), plotX1 - 5, y);
28    }
29  }
30
31
32
33  void drawAxisLabels( )
34  {
35    fill(0);
36    textSize(13);
37    textLeading(15);
38    textAlign(CENTER, CENTER);
39    // Use \n (aka enter or linefeed) to break the text into separate lines.
40    text("Salida\ndel\nsistema", labelX, (plotY1+plotY2)/2);
41    textAlign(CENTER);
42    text("Instante de muestreo", (plotX1+plotX2)/2, labelY);
43  }
44
45  void drawTitle()
46  {
47    fill(0);
48    textSize(25);
49    String title = "Gráfica de salida del sistema";
50    text(title , plotX1+700, plotY1 - 30);
51  }
52
53  void MaxAndMin(float valor)
54  {
55    if (valor<dataMin)
```

```
56    {
57      dataMin=valor;
58    }
59    if (valor>dataMax)
60    {
61      dataMax=valor;
62    }
63    intervaloY=(dataMax-dataMin)/10;
64  }
```

Capítulo 15

Anexo 7: Código fuente Fuente PID Matlab

En el presente anexo se incluyen los cuatro módulos que componen el programa utilizado para realizar un ejemplo completo de utilización del equipo (sección 8.9). Los cuatro módulos son: módulo principal, módulo de configuración del sistema, módulo de configuración del regulador y el módulo de visualización de los datos

15.1. Módulo principal

```
1  %Configuración del sistema (sólo la primera ejecución)
2  system_config
3  %Configuración del controlador (cada vez que se quiera cambiar algo en él)
4  system_controller
5  %Inicialización de variables
6  Error=zeros(size_num,1); %Inicialización del vector de error (el tamaño
7  %depende del numerador del regulador)
8  Control_signal=zeros(size_den,1); %Inicialización del vector de la señal
9  %de control (el tamaño depende del denominador del regulador)
10 output_value=0; %Inicialización del valor de salida
11 input_value=0; %Inicialización del valor de entrada
12 Consigna=30; %Inicialización de la consigna
13 all_data=zeros(n,4); %Se almacena el tiempo, la consigna, la salida
14 %real y la señal de control
15 all_data(:,2)=Consigna; %La consigna será siempre la misma.
16 tic; %Activación del temporizador
17
18 %Bucle principal de la prueba
19 for i=1:n
20     start_time=toc; %Almacena el tiempo de inicio de la muestra
```

```
21      Arduino_Uno.analogWrite(5,output_value); %Genera la tensión de salida
22      input_value=Arduino_Uno.analogRead(3); %Adquiere el tensión de entrada
23      %Instrucciones de acondicionamiento de entrada
24      input_voltage=(input_value*5)/1023; %Tensión en la entrada
25      Salida_real=(input_voltage*100)/5; %Cálculo de la salida real
26      %Instrucciones para calcular la señal de control con el regulador PID
27      Error(2:size_num)=Error(1:size_num-1); %Se desplazan los valores del
28      %vector para ordenador según se adquirieron
29      Error(1)=Consigna-Salida_real; %Se actualiza el vector con el nuevo
30      %error de la esta muestra
31      Control_signal(2:size_den)=Control_signal(1:size_den-1); %Se desplazan
32      %los valores del vector para ordenarlos según se adquirieron
33      %Calculo de la nueva señal de control según ecuaciones de diferencias
34      Control_signal(1)=(Num_PID*Error-Den_PID(2:size_den)*Control_signal(2:
            size_den))/Den_PID(1);
35
36      %Ajuste de la señal si sobrepasa los límites permitidos (0:100)
37      if Control_signal(1)>100
38          Control_signal(1)=100;
39      elseif Control_signal(1)<0
40          Control_signal(1)=0;
41      end
42      %Instrucciones de acondicionamiento de salida
43      output_voltage=Control_signal(1)/20; %Cálculo de la tensión de salida
44      output_value=round((output_voltage*255)/5); %Escalado de la salida
45      %Vuelve a calcular la señal de control. Debido al redondeo anterior
46      Control_signal(1)=output_value*5*20/255;
47      %Instrucciones de almacenamiento de resultados
48      all_data(i,1)=i*T; %Se almacena el instante de muestreo
49      all_data(i,3)=Salida_real; %Se almacena la temperatura real
50      all_data(i,4)=Control_signal(1); %Se almacena la señal de control
51      %Instrucción para generar una gráfica en tiempo real
52      if(rem(i,coef_graph)==0)
53          system_graph(all_data,n_graph);
54      end
55      %Instrucciones para comprobar que se cumplió el tiempo de muestreo
56      spend_time=toc-start_time; %Se comprueba si el tiempo transcurrido
57      %permite cumplir con el período de muestreo
58      if spend_time>T
59          disp('Tiempo de muestreo superado'); %Se muestra la causa de fallo
60          break; %Si el tiempo es mayor aborta la muestra
61      else
62          wait_time=T-spend_time; %Si el tiempo transcurrido es menor, se
63          pause(wait_time); %esperará hasta completar T
64      end
```

```
65 end
66 %Reseteo de salidas
67 Arduino_Uno.analogWrite(5,0); %Desactiva la salida al finalizar
68
69 %Presentación de resultados
70 figure
71 system_graph(all_data,n);
72 save data
```

15.2. Módulo de configuración del sistema

```
1  clear
2  clc
3  %Definiciones de la parte de muestras muestreo
4  T=0.1; %Período de muestreo
5  total_time=500; %Tiempo total de la prueba
6  n=round(total_time/T); %Número de muestras en la prueba
7  %Instrucciones de configuración del Arduino
8  Arduino_Uno=arduino('COM14');
9  Arduino_Uno.pinMode(5,'output');
10 %Configuración de la gráfica
11 graphic_time=120; %Tiempo a representar en la gráfica
12 coef_graph=2; %Cada cuantas muestras se actualiza la gráfica
13 n_graph=round(graphic_time/T); %Número de datos que se van a representar
```

15.3. Módulo de configuración del regulador

```
1  %Configuración del controlador
2  K=1.1284; %Ganancia proporcional del regulador PID
3  Ti=0.5017; %Tiempo integral del regulador PID
4  Td=0.3345; %Tiempo derivativo del regulador PID
5
6  G_PID=K*tf([Ti*Td Ti 1],[Ti 0]); %Se crea la función de transferencia del
7  %regulador
8  G_PID=c2d(G_PID,T,'tustin'); %Se discretiza la función de transferencia
9
10 Num_PID=G_PID.num{1}; %Se queda con los datos del numerador
11 Den_PID=G_PID.den{1}; %y del denominador
12
13 size_num=size(Num_PID,2); %Obtiene el tamaño del numerador (tamaño del
14 %vector de error)
```

```
15 size_den=size(Den_PID,2); %Obtiene el tamaño del numerador (tamaño del
16 %vector de la señal de control)
```

15.4. Módulo de visualización de los datos

```
1  function system_graph(all_data,n_graph)
2      %Instrucciones para la presentación de la gráfica
3      n=sum(0~=all_data(:,1)); %Comprueba el número de muestras válidas
4      %(las que tienen un tiempo diferente de 0)
5      if (n<n_graph) %Si no hay suficientes muestras como para presentar los
6      n_graph=n; %puntos solicitados, presenta todos los datos
7      end
8      all_data=all_data(1+(n-n_graph):n,:); %Se queda con datos válidos
9      clf %Se borra la gráfica que hubiera
10     %Representa las tres magnitudes grabadas (la consigna, la salida
11     %real, y la señal de control)
12     plot(all_data(:,1),all_data(:,2),all_data(:,1),all_data(:,3));
13     if n>1
14     axis([min(all_data(:,1)) max(all_data(:,1)) min(min(all_data(:,2:4)))
           max(max(all_data(:,2:4)))]);
15     end
16 end
```

PLANOS

TÍTULO: **DESARROLLO Y TESTEO DE UN EMULADOR DE PLANTAS INDUSTRIALES BASADO EN ARDUINO**

PLANOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

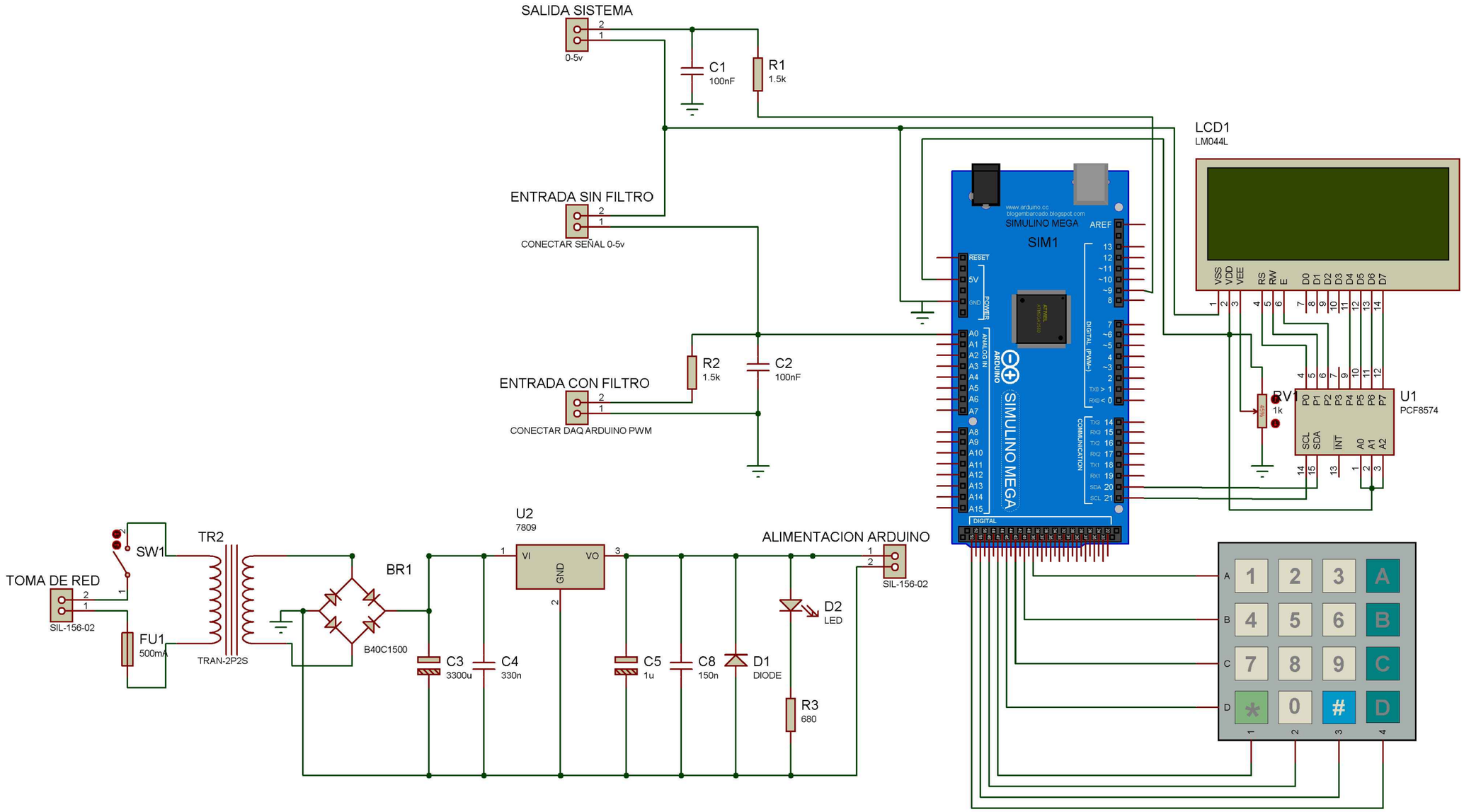
15405 - FERROL

FECHA: JUNIO DE 2014

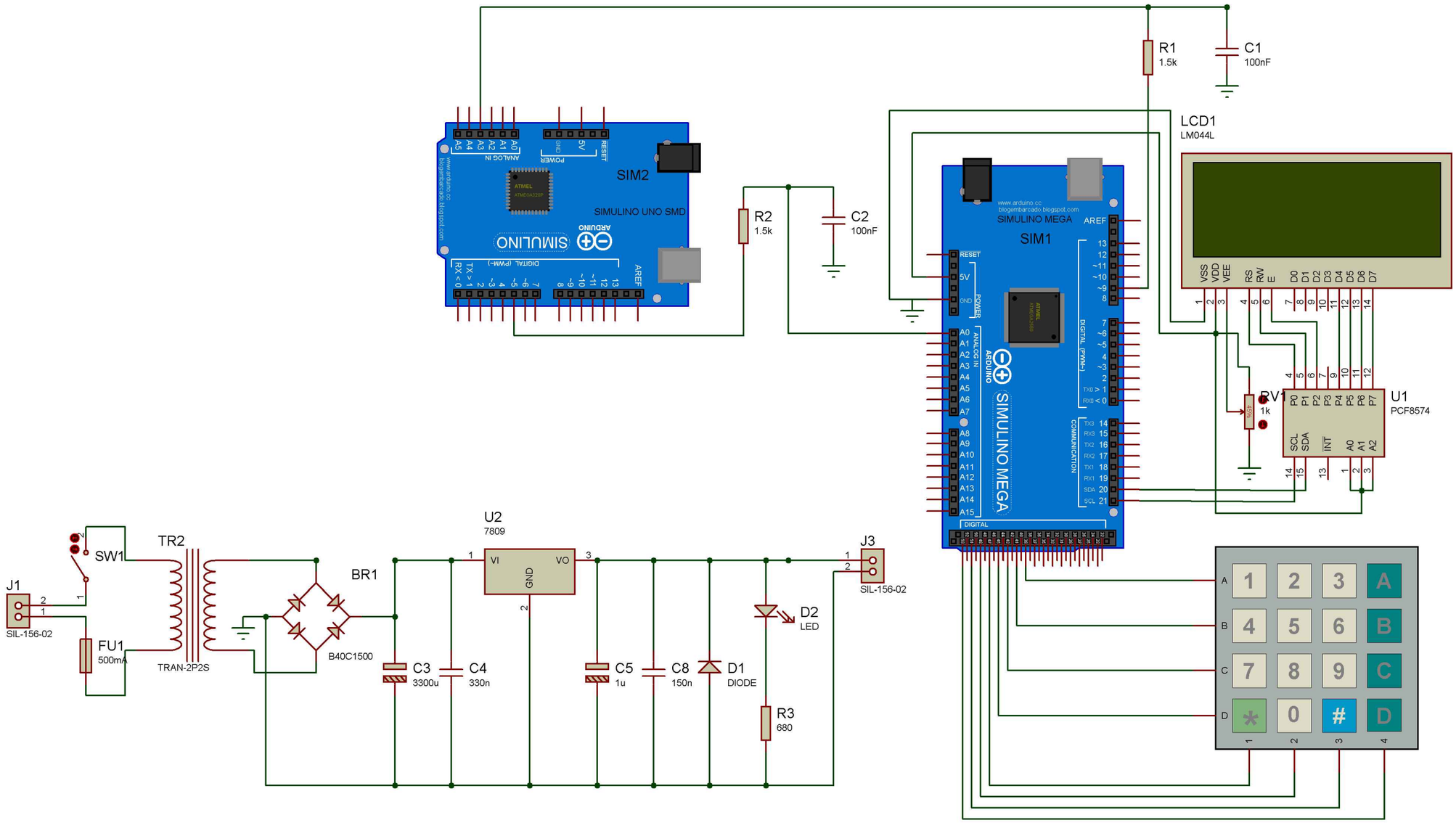
AUTOR: ESTEBAN VELO SÁNCHEZ

Fdo.:

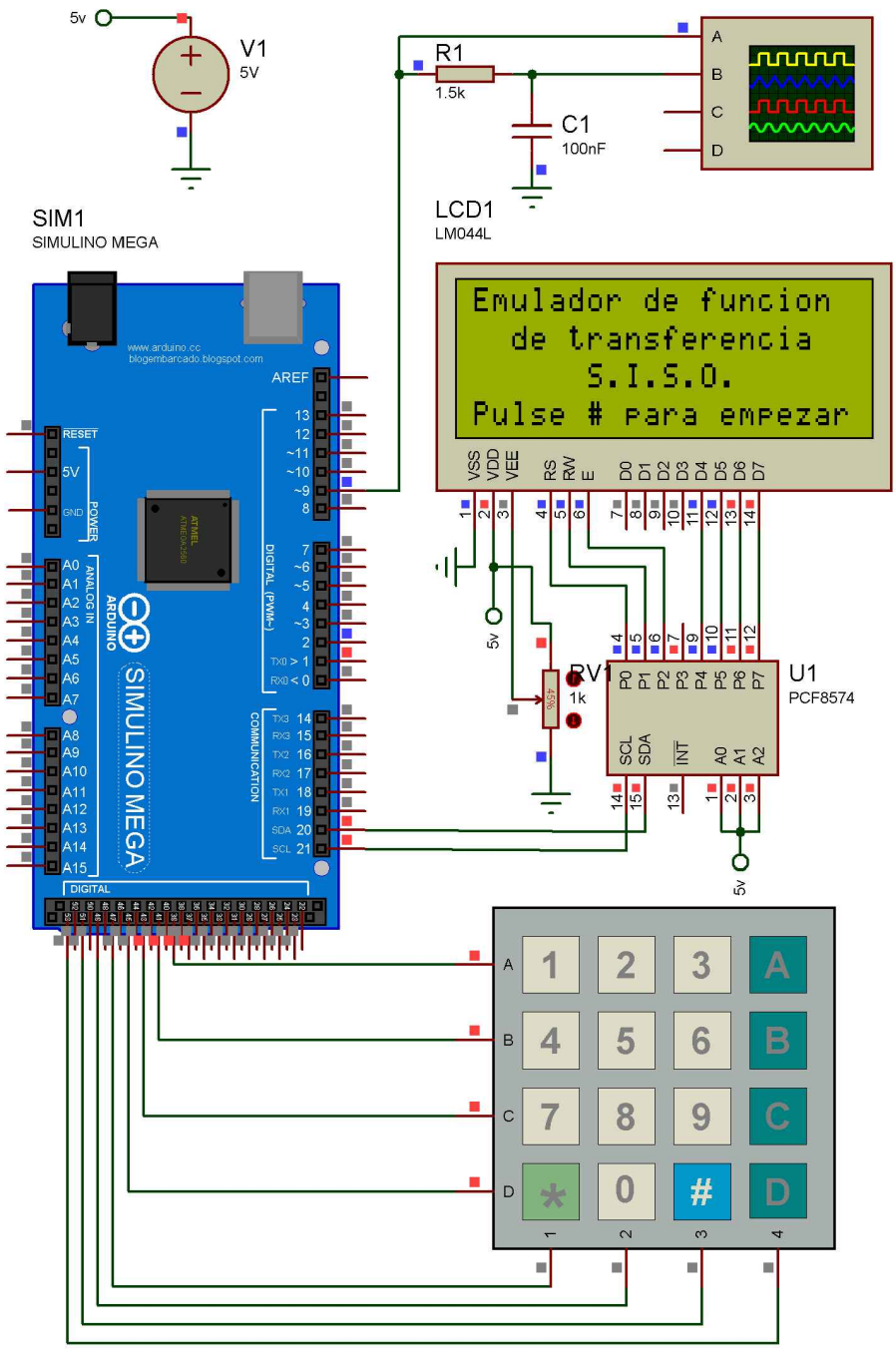
Índice de planos



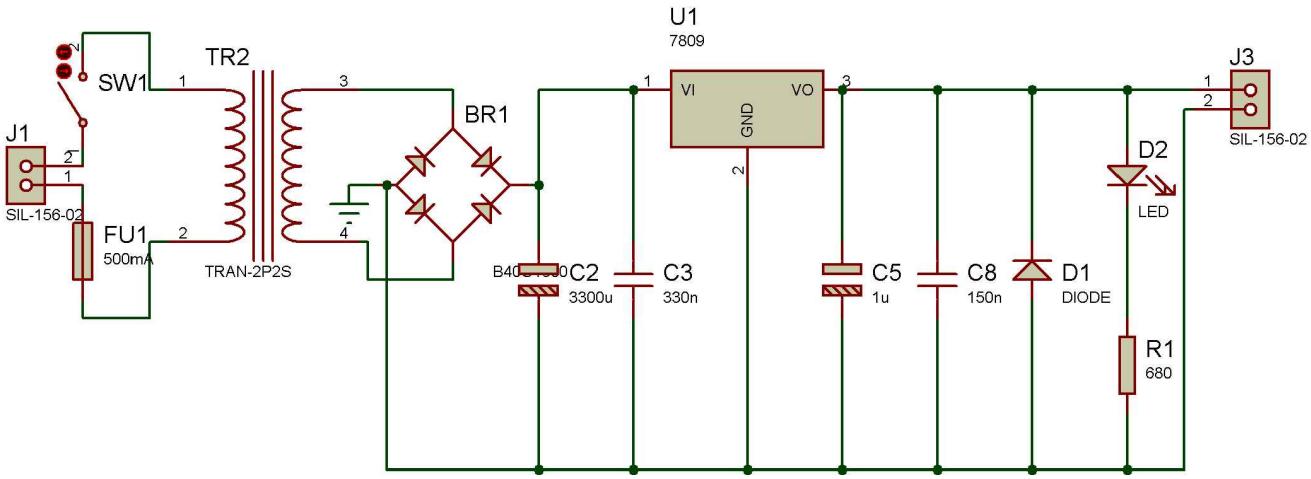
UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA	TFG Nº: 770G01A52
TÍTULO DEL TFG: EMULADOR DE PLANTAS INDUSTRIALES	
TÍTULO DEL PLANO: EQUIPO	FECHA: JUNIO 2014 ESCALA: ***
AUTOR: ESTEBAN VELO SÁNCHEZ FIRMA:	PLANO Nº: 1



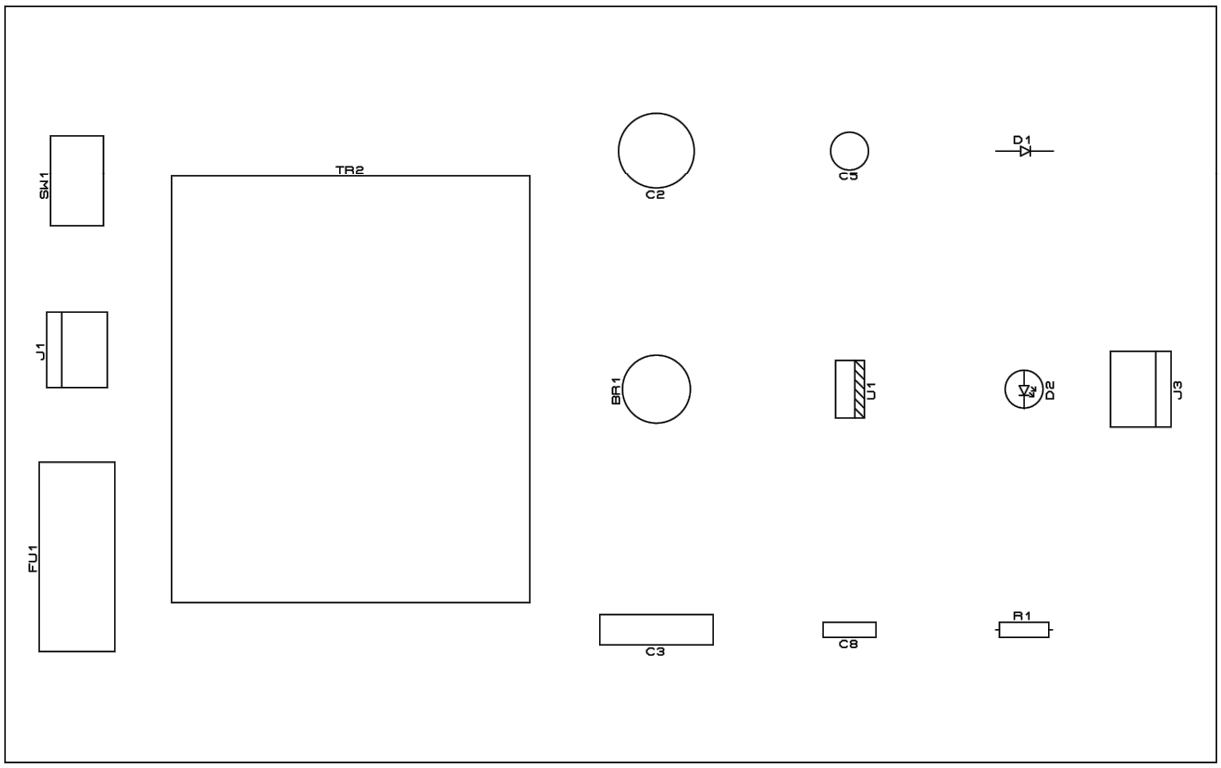
UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA		TFG Nº: 770G01A52
TÍTULO DEL TFG: EMULADOR DE PLANTAS INDUSTRIALES		
TÍTULO DEL PLANO: CONEXIÓN ARDUINO EXTERNO		FECHA: JUNIO 2014 ESCALA: ***
AUTOR: ESTEBAN VELO SÁNCHEZ	FIRMA:	PLANO Nº: 2



UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA	TFG Nº: 770G01A52
TÍTULO DEL TFG: EMULADOR DE PLANTAS INDUSTRIALES	
TÍTULO DEL PLANO: SIMULACIÓN	FECHA: JUNIO 2014 ESCALA: ***
AUTOR: ESTEBAN VELO SÁNCHEZ	FIRMA:
	PLANO Nº: 3



UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA	TFG Nº: 770G01A52
TÍTULO DEL TFG: EMULADOR DE PLANTAS INDUSTRIALES	
TÍTULO DEL PLANO: CIRCUITO DE LA FUENTE DE ALIMENTACIÓN	FECHA: JUNIO 2014 ESCALA: ***
AUTOR: ESTEBAN VELO SÁNCHEZ FIRMA:	PLANO Nº: 4



UNIVERSIDADE DA CORUÑA

ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A52

TÍTULO DEL TFG:

EMULADOR DE PLANTAS INDUSTRIALES

TÍTULO DEL PLANO:

CARA COMPONENTES FUENTE

FECHA: JUNIO 2014

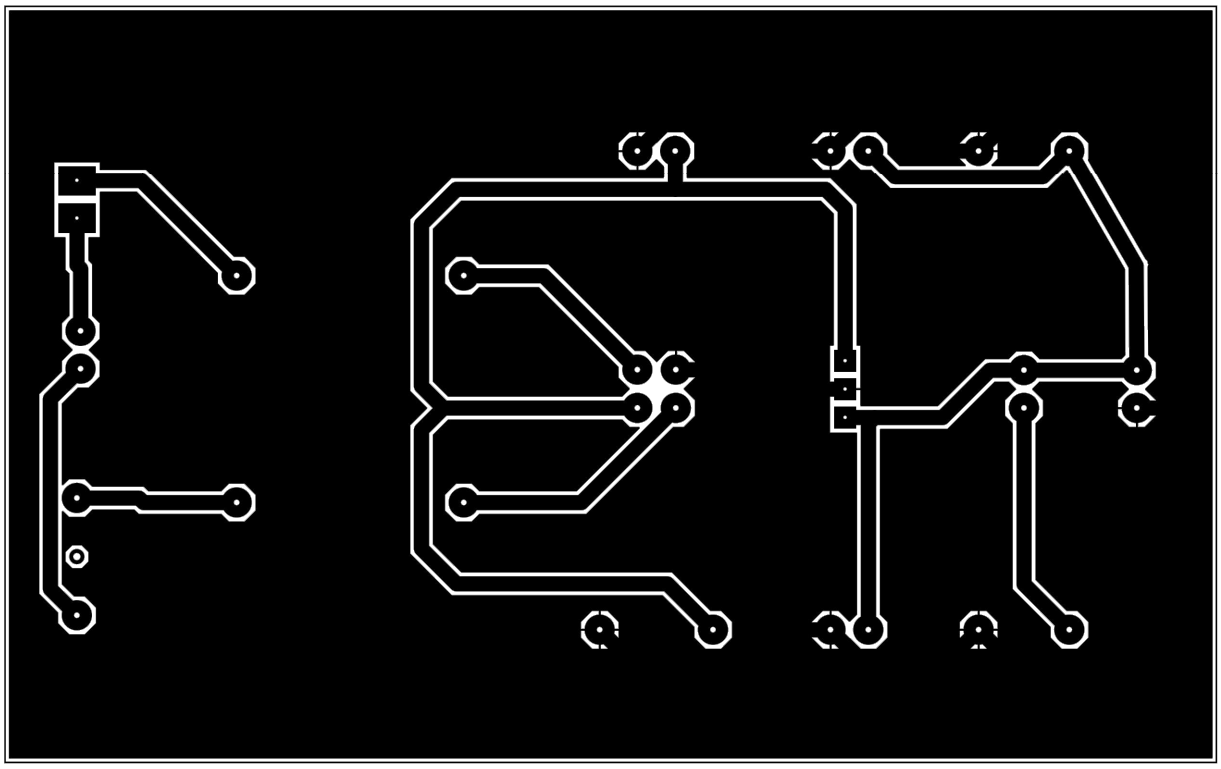
ESCALA: 1:1

AUTOR:

ESTEBAN VELO SÁNCHEZ

FIRMA:

PLANO Nº: 5



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A52

TÍTULO DEL TFG:

EMULADOR DE PLANTAS INDUSTRIALES

TÍTULO DEL PLANO:

CARA PISTAS FUENTE

FECHA: JUNIO 2014

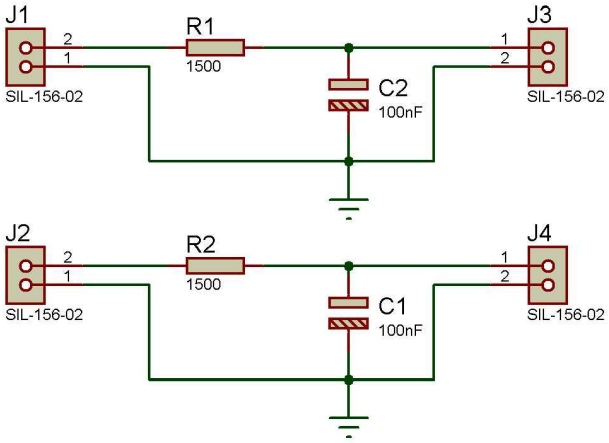
ESCALA: 1:1

AUTOR:

ESTEBAN VELO SÁNCHEZ

FIRMA:

PLANO Nº: 6



UNIVERSIDADE DA CORUÑA

ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A52

TÍTULO DEL TFG:

EMULADOR DE PLANTAS INDUSTRIALES

TÍTULO DEL PLANO:

CIRCUITO DE ACONDICIONAMIENTO

FECHA: JUNIO 2014

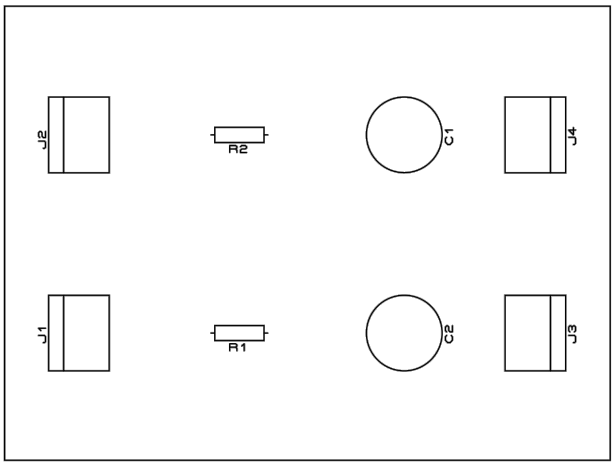
ESCALA: ***

AUTOR:

ESTEBAN VELO SÁNCHEZ

FIRMA:

PLANO Nº: 7



UNIVERSIDADE DA CORUÑA

ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A52

TÍTULO DEL TFG:

EMULADOR DE PLANTAS INDUSTRIALES

TÍTULO DEL PLANO:

CARA COMPONENTES ACONDICIONAMIENTO

FECHA: JUNIO 2014

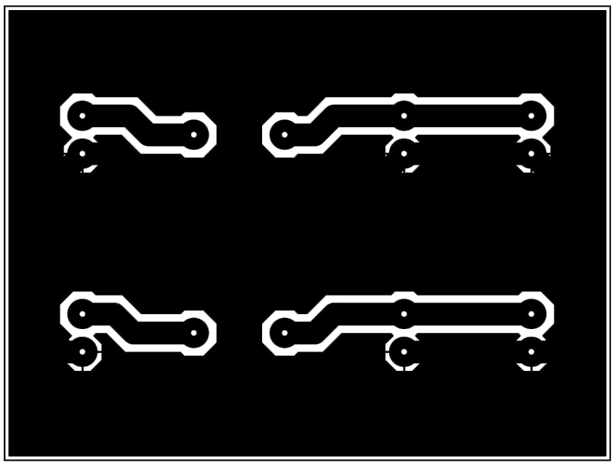
ESCALA: 1:1

AUTOR:

ESTEBAN VELO SÁNCHEZ

FIRMA:

PLANO Nº: 8



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A52

TÍTULO DEL TFG:

EMULADOR DE PLANTAS INDUSTRIALES

TÍTULO DEL PLANO:

CARA PISTAS ACONDICIONAMIENTO

FECHA: JUNIO 2014

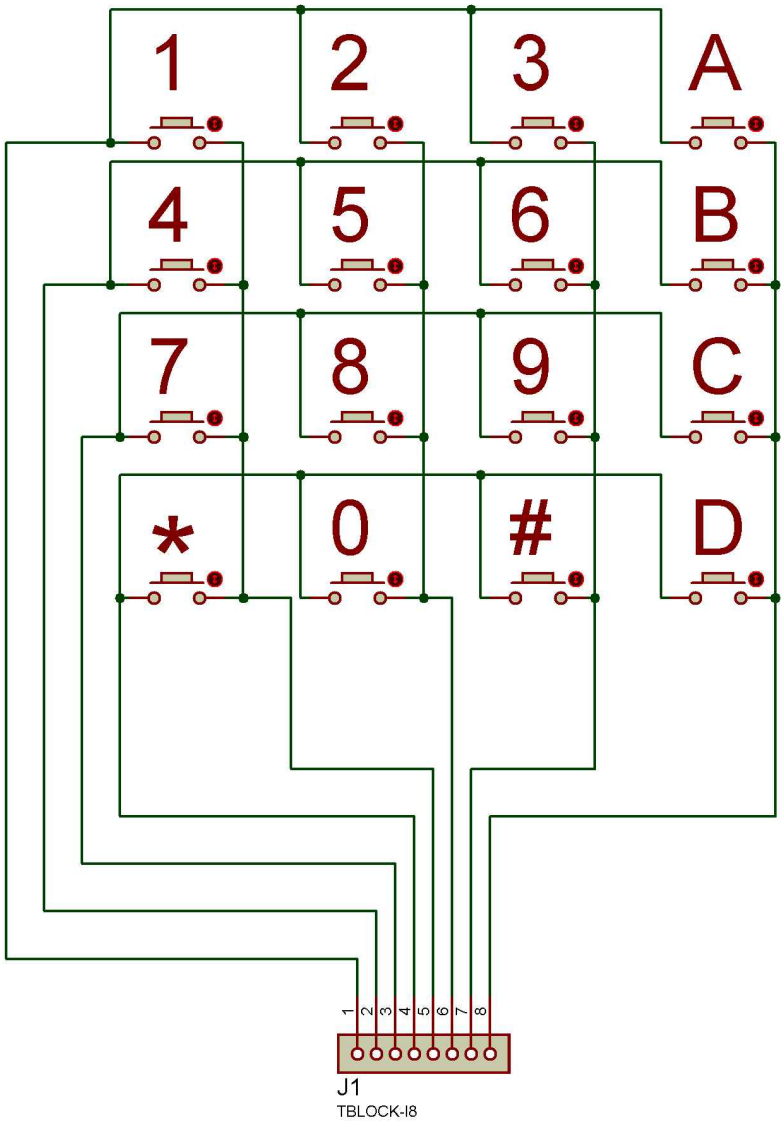
ESCALA: 1:1

AUTOR:

ESTEBAN VELO SÁNCHEZ

FIRMA:

PLANO Nº: 9



UNIVERSIDADE DA CORUÑA

ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A52

TÍTULO DEL TFG:

EMULADOR DE PLANTAS INDUSTRIALES

TÍTULO DEL PLANO:

DETALLE TECLADO

FECHA: JUNIO 2014

ESCALA: ***

AUTOR:

ESTEBAN VELO SÁNCHEZ

FIRMA:

PLANO Nº: 10

PLIEGO DE CONDICIONES

TÍTULO: **DESARROLLO Y TESTEO DE UN EMULADOR DE PLANTAS INDUSTRIALES BASADO EN ARDUINO**

PLIEGO DE CONDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

15405 - FERROL

FECHA: JUNIO DE 2014

AUTOR: ESTEBAN VELO SÁNCHEZ

Fdo.:

Capítulo 16

Especificaciones de los materiales

Los materiales y componentes utilizados en la realización del presente proyecto han sido aquellos que reunieran los requisitos de diseño. Además éstos debían asegurar un correcto funcionamiento durante la vida del dispositivo.

El subministrador debe garantizar el funcionamiento correcto de cada componente según las hojas características del mismo. Además debe proporcionar las distintas pruebas que le debe realizar antes de su utilización en el dispositivo emulador.

Capítulo 17

Pruebas de verificación

A continuación se detallan las pruebas de verificación necesarias para comprobar el correcto funcionamiento de cada uno de los módulos que componen el dispositivo emulador.

17.1. Fuente de alimentación

Se deben comprobar las tensiones en cada etapa de la fuente de alimentación.

En primer lugar se debe comprobar la tensión de la línea: Ésta debe ser cercana a los 230 V y la frecuencia de 50Hz.

En segundo lugar se comprobará que la tensión de salida del transformador es de los 12v indicados por el fabricante y que la frecuencia sigue siendo 50Hz.

A continuación se comprueba si el rectificar hace su función, tan sólo se deben ver los semiciclos positivos a los que se debe restar la tensión de caida en dos diodos (dato proporcionado por el fabricante).

El condensador de filtro debe filtrar la tensión hasta obtener una tensión continua con un pequeño rizado.

Para finalizar, se debe comprobar que la tensión de salida de la fuente de alimentación son los 9v necesarios.

17.2. Circuito de acondicionamiento

Deben comprobarse todas las conexiones y el funcionamiento correcto del circuito. Para se debe introducir una señal cuadrada de prueba a la frecuencia de 32 kHz y a la salida se debe obtener una tensión igual a la mitar de la tensión de pico de entrada introducida.

17.3. Teclado

La comprobación del teclado puede realizarse midiendo continuidad entre la fila y la columna correspondientes a una tecla pulsada. Esto debe hacerse para todas las teclas.

17.4. LCD

Se debe comprobar que el LCD enciende correctamente con tan sólo introducirle la tensión de alimentación. Se recomienda realizar pruebas de que funcionan las cuatro líneas disponibles.

17.5. Arduino

Debe comprobarse que el Arduino utilizado está en perfectas condiciones, se recomiendo probar las salidas que se van a utilizar. También se debe comprobar el regulador de tensión de 5v de Arduino.

Capítulo 18

Condiciones de almacenamiento

El equipo debe estar protegido de la luz solar para evitar daños en el LCD. La temperatura de almacenamiento debe ser inferior a 35°C. El ambiente debe ser seco y con ausencia de polvo.

Capítulo 19

Guía de implementación

En la última parte del pliego se elaboran las condiciones de implementación del equipo. Cuando se pretenda realizar el montaje del mismo debe cumplirse esta guía para que el montaje sea correcto.

19.1. Primeros pasos

Antes de comenzar a programar el Arduino se deben instalar los drivers del mismo y el software de programación adecuados. Esto se puede descargar desde la página oficial de Arduino, en el siguiente enlace:

http://arduino.cc/en/Main/Software

En este enlace hay que selecionar el paquete disponible acorde al sistema operativo que se tenga instalado en el ordenador donde se va a usar el Arduino. Se recomienda descargar la versión de Arduino IDE 1.0.5 en su versión comprimida en zip (figura 19.1.0.1). De esta forma se asegura la compatibilidad de todas las librerías utilizadas durante la fase de realización del proyecto.

Para instalar los drivers correctamente se debe conectar el Arduino Mega 2560 al PC por medio del cable USB (tipo A-B). Cuando el sistema operativo solicite los drivers habrá que indicarle que se encuentran en una carpeta especificada. Esta carpeta será la que se acaba de descargar anteriormente en formato ZIP y que previamente haya descomprimido. Cuando pida la ubicación de los drivers se debe seleccionar la carpeta que se muestra en la figura 19.1.0.2:

Figura 19.1.0.1 – Arduino 1.0.5

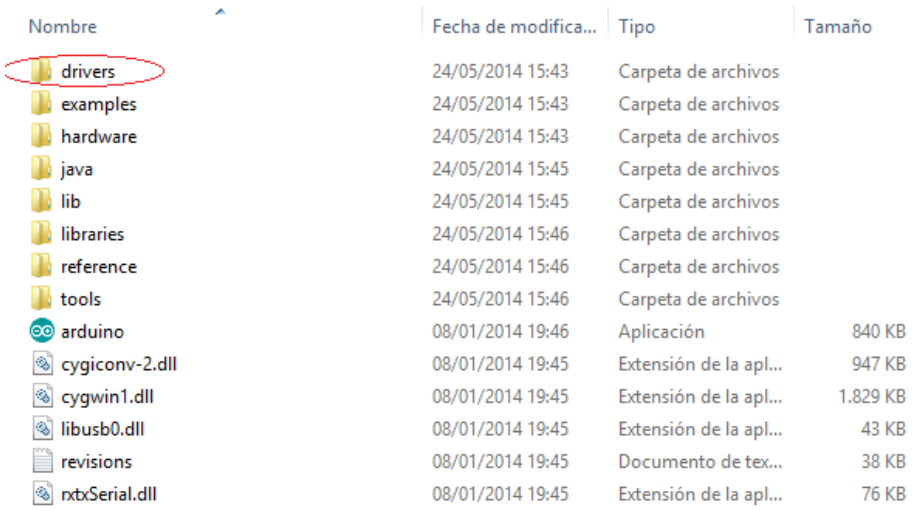


Figura 19.1.0.2 – Driver

Seguidamente se presiona el botón de instalar que aparecerá en una ventana similar a la que se muestra en la figura 19.1.0.3:

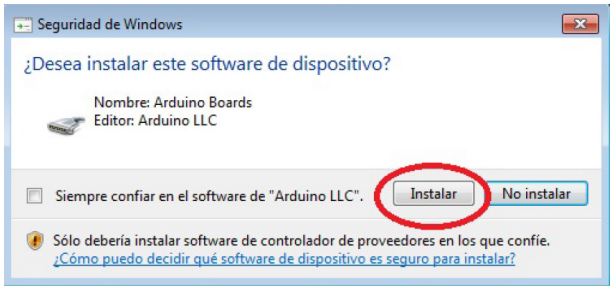


Figura 19.1.0.3 – Instalar

Se espera a que termine y a continuación se comprueba que los drivers quedaran correctamente instalados. Para ello se debe acceder al Administrador de Dispositivos del PC y comprobar que el Arduino aparezca en la pestaña de puertos COM del equipo como en la captura mostrada en la figura 19.1.0.4 (el número del puerto COM asignado puede ser diferente):

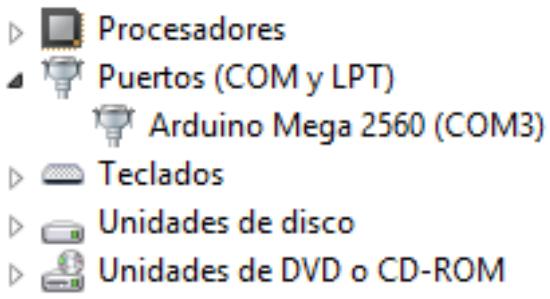


Figura 19.1.0.4 – COM

Una vez en este punto ya se está en disposición de programar el Arduino con el código fuente necesario para la simulación de las funciones de transferencia. Pero antes de ello, se debe comprobar que tiene instaladas las librerías de programación necesarias: La librería de control del teclado y la de control del LCD. La carpeta *libraries* quedaría como la de la figura 19.1.0.5.

Figura 19.1.0.5 – Librerias

Una vez hecho esto el software y el hardware de Arduino están preparados para programarlos de forma que los periféricos conectados funcionen de forma correcta y con una programación sencilla y muy intuitiva. Lo que se debe hacer a continuación será compilar y cargar el archivo *fuente.ino* con todo el código fuente en el Arduino Mega.

19.2. Montaje de circuitos

Para el montaje de la totalidad del equipo existen tres circuitos principales:

- Circuito del Arduino, LCD y teclado. Éste se corresponde con el plano [1].
- Circuito de la fuente de alimentación del equipo (placa de circuito impreso). Éste se corresponde con el plano [4].
- Circuito de la placa de acondicionamiento (placa de circuito impreso). Éste se corresponde con el plano [7].

Para el montaje del primer circuito se utilizarán cables con terminales crimpados para que no se suelten. Se deberá seguir el esquema reflejado en el circuito del plano [1]. La longitud de los cables de conexiones debe ser de al menos 20 cm para dejar holgura suficiente. De esta forma el montaje posterior del equipo será mucho más sencillo.

Para el montaje de la fuente de alimentación se deberá realizar la placa de circuito impreso que está plasmada en el plano [6]. El plano [5] indica donde va colocado cada componente. Se debe realizar el montaje basándose en este plano y el circuito de la fuente de alimentación (plano [4]).

Para el montaje de la placa de acondicionamiento se deberá implementar la placa de circuito impreso del plano [9]. El plano [8] indica donde va colocado cada componente. Se debe realizar el montaje basándose en este plano y el circuito del plano [7].

Las placas de circuito impreso diseñadas corresponden con los componentes mencionados en el Estado de Mediciones. Los dos planos están a escala real. De esta forma se pueden imprimir directamente en papel transparente para poder insolarlos posteriormente.

19.3. Conexionado de circuitos

Se debe consultar el circuito general del equipo [1] para realizar la conexión de los distintos circuitos implementados anteriormente. La fuente de alimentación sólo alimentará el Arduino ya que el resto de tensiones se obtienen de él. Aunque en los circuitos no aparezca, la tensión de salida de la fuente de alimentación se debe conectar a las patillas Vin y GND de alimentación externa del Arduino.

19.4. Montaje final en caja

Una vez conexionado todo el equipo, se debe realizar su montaje en una caja. Los distintos circuitos irán atornillados a ella de forma que queden fijos. La caja llevará un interruptor externo de encendido del equipo y un agujero para pasar el cable de alimentación del equipo hacia el interior de la caja. Además deberá permitir acceder al puerto USB del Arduino al exterior de la caja. La caja debe contar con las siguientes conexiones:

- Un bornero de tres entradas: Entrada (0-5v), entrada PWM (0-5v) y masa.
- Un bornero de dos salidas: Salida (0-5v) y masa.

La entrada PWM es la que se denomina en el plano general del equipo [1] como entrada sin filtro. Se debe asegurar de que todas las masas estén conectadas de forma que no haya ningún punto flotante.

El teclado y el display LCD se montarán en la parte superior de la caja. Se deben colocar de forma que el teclado quede inmediatamente debajo del display LCD. El display irá fijado a la placa por medio de cuatro tornillos, uno en cada una de las esquinas. El teclado simplemente irá pegado con un adhesivo que garantice la unión.

ESTADO DE MEDICIONES

TÍTULO: **DESARROLLO Y TESTEO DE UN EMULADOR DE PLANTAS INDUSTRIALES BASADO EN ARDUINO**

ESTADO DE MEDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

15405 - FERROL

FECHA: JUNIO DE 2014

AUTOR: ESTEBAN VELO SÁNCHEZ

Fdo.:

Capítulo 20

Dispositivos del equipo

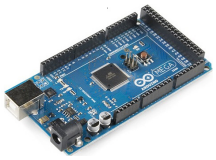
ID	Imagen	Descripción	Ref. RS	Ref.Farnell	Ud
SIM		Arduino Mega 2560 REV 3 A000067	715-4084	2212779	1
LCD1		Display LCD 20x4 LCD 2004	720-0226	2342648	1
I2C		modulo i2c para display LCD	————	————	1
KEY		Teclado matricial de membrana 4x4	507-497	1774831	1

Tabla 20.0.0.1 – Lista de materiales 1

Capítulo 21

Fuente de alimentación

ID	Imagen	Descripción	Ref. RS	Ref.Farnell	Ud
U2		Regulador de tensión lineal, LM7809ACT, 9V, 1A, TO-220	806-2845	1126247	1
BR1		Puente rectificador B40-C1500 40v 1,5A	700-5386	1336498	1
TR2		Transformador Myrra 44302 9V 16VA	————	1689093	1
FU1		Portafusibles 5x20mm PCB	787-4164	146123	1
FUS		Fusible 5x20mm 500mA 250v	537-1240	1354567	1

Tabla 21.0.0.1 – Lista de materiales 2

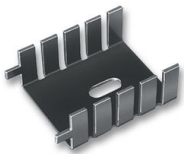
ID	Imagen	Descripción	Ref. RS	Ref.Farnell	Ud
C8		Condensador cerámico 150nF 50v	699-5156	2112907	1
DIS		Disipador para T0-220 20°C/W	189-9132	4621281	1
SW1		Interruptor ON-OFF 250v	783-1135	1661841	1
JX		Conector PCB 2P	803-2844	3882573	2
D1		Diodo de protección 1n4001 1A	774-3265	1843694	1
D2		Diodo led verde 3mm	708-2759	2373465	1
R3		Resistencia 1/4w 680Ω	707-7656	2368815	1
PCB		Placa de circuito impreso fotosensible 160x100mm	159-6057	1267751	1

Tabla 21.0.0.2 – Lista de materiales 3

ID	Imagen	Descripción	Ref. RS	Ref.Farnell	Ud
C3		Condensador electrolítico radial de aluminio 3300 μF 16v	711-1019	2346506	1
C5		Condensador electrolítico radial de aluminio 1 μF 50v	711-1407	1167954	1
C4		Condensador cerámico 330nF 50v	721-5278	1100395	1

Tabla 21.0.0.3 – Lista de materiales 6

Capítulo 22

Placa de acondicionamiento

ID	Imagen	Descripción	Ref. RS	Ref.Farnell	Ud
JX		Conector PCB 2P	803-2844	3882573	4
R1,R2		Resistencia 1/4w 1500Ω	707-7681	1127991	2
C1,C2		Condensador electrolítico 100nF 50v	520-1236	1902061	2
PCB2		Placa de circuito impreso fotosensible 80x60mm	————	————	1

Tabla 22.0.0.1 – Lista de materiales 4

Capítulo 23

Tarjeta de adquisición de datos

ID	Imagen	Descripción	Ref. RS	Ref.Farnell	Ud
SIM2		Arduino Leonardo Rev 3 A000057	761-7324	2133071	1

Tabla 23.0.0.1 – Lista de materiales 5

El componente I2C y el PCB2 no se encuentran en Farnell ni en RS pero se puede encontrar en cualquier otra tienda de electrónica online.

PRESUPUESTO

TÍTULO: **DESARROLLO Y TESTEO DE UN EMULADOR DE PLANTAS INDUSTRIALES BASADO EN ARDUINO**

PRESUPUESTO

PETICIONARIO: **ESCOLA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBRERO, S/N

15405 - FERROL

FECHA: JUNIO DE 2014

AUTOR: ESTEBAN VELO SÁNCHEZ

Fdo.:

Capítulo 24

Presupuesto de materiales

24.1. Presupuesto de dispositivos

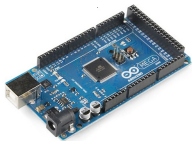
ID	Imagen	Descripción	PVP RS	PVP Farnell	Ud	Total
SIM1		Arduino Mega 2560 REV 3 A000067	43,06 €	34,27 €	1	34,27 €
LCD1		Display LCD 20x4 LCD 2004	11,94 €	13,59 €	1	11,94 €
I2C		modulo i2c para display LCD	———-	————	1	1,49 €
KEY		Teclado matricial de membrana 4x4	14,21 €	15,63 €	1	14,21 €
					TOTAL:	61,94 €

Tabla 24.1.0.1 – Presupuesto de dispositivos

24.2. Presupuesto fuente de alimentación

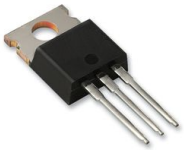
ID	Imagen	Descripción	PVP RS	PVP Farnell	Ud	Total
U2		Regulador de tensión lineal, LM7809ACT, 9V, 1A, TO-220	0,31 €	0,40 €	1	0,31 €
BR1		Puente rectificador B40-C1500 40v 1,5A	0,39 €	0,35 €	1	0,35 €
TR2		Transformador Myrra 44302 9V 16VA	———	9,39 €	1	9,39 €
FU1		Portafusibles 5x20mm PCB	2,92 €	0,76 €	1	0,76 €
FUS		Fusible 5x20mm 500mA 250v	0,27 €	0,25 €	1	0,25 €
C3		Condensador electrolítico radial de aluminio 3300 μF 16v	0,43 €	0,9 €	1	0,43 €
C5		Condensador electrolítico radial de aluminio 1 μF 50v	0,027 €	0,059 €	1	0,027 €
C4		Condensador cerámico 330nF 50v	0,45 €	0,79 €	1	0,45 €

Tabla 24.2.0.2 – Presupuesto fuente de alimentación 1

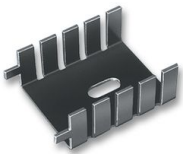
ID	Imagen	Descripción	PVP RS	PVP Farnell	Ud	Total
C8		Condensador cerámico 150nF 50v	1,64 €	0,27 €	1	0,27 €
DIS		Disipador para T0-220 20°C/W	0,61 €	1,22 €	1	0,61 €
SW1		Interruptor ON-OFF 250v	12,43 €	3,63 €	1	3,63 €
JX		Conector PCB 2P	1,33 €	0,64 €	2	1,28 €
D1		Diodo de protección 1n4001 1A	0,04 €	0,13 €	1	0,04 €
D2		Diodo led verde 3mm	0,22 €	0,11 €	1	0,11 €
R3		Resistencia 1/4w 680Ω	0,02 €	0,01 €	1	0,01 €
PCB		Placa de circuito impreso fotosensible 160x100mm	4,23 €	3,39 €	1	3,39 €
					TOTAL:	21,31 €

Tabla 24.2.0.3 – Presupuesto fuente de alimentación 2

24.3. Presupuesto placa de acondicionamiento

ID	Imagen	Descripción	PVP RS	PVP Farnell	Ud	Total
JX		Conector PCB 2P	1,33 €	0,64 €	4	2,56 €
R1,R2		Resistencia 1/4w 1500Ω	0,02 €	0,01 €	2	0,02 €
C1,C2		Condensador electrolítico 100nF 50v	0,18 €	0,07 €	2	0,14 €
PCB2		Placa de circuito impreso fotosensible 80x60mm	———	———	1	3,54 €
				TOTAL:		6,26 €

Tabla 24.3.0.4 – Presupuesto de la placa de acondicionamiento

24.4. Presupuesto de la tarjeta de adquisición de datos

ID	Imagen	Descripción	PVP RS	PVP Farnell	Ud	Total
SIM2		Arduino Leonardo Rev 3	17,63 €	21,59 €	1	17,63 €
				TOTAL:		17,63 €

Tabla 24.4.0.5 – Presupuesto de la DAQ

Capítulo 25

Presupuesto de Recursos Humanos

Concepto	Nº de horas	Precio/hora	TOTAL
Ingeniería	300	60 €	18000 €
Montaje	16	40 €	640 €
		TOTAL:	18640 €

Tabla 25.0.0.1 – Presupuesto RRHH

Capítulo 26

Presupuesto final

Concepto	SUBTOTAL
Ingeniería	18000 €
Montaje	640 €
Material fuente	21,31 €
Material filtro	6,26 €
Dispositivos	61,94 €
Material montaje	50 €
DAQ	17,63 €
IVA (21 %)	3947,4 €
TOTAL:	22744,5 €

Tabla 26.0.0.1 – Presupuesto TOTAL

El presupuesto total del proyecto asciende a la cantidad de: ***Veintidos mil setecientos cuarenta y cuatro con cinco euros*** (22744,5 €).